2013

Manual SDK Acesso e Ponto

Manual do Desenvolvedor

Versão 3.1 - 20/04/2013

Digicon S.A
2013

Protocolo: 3
Revisão: 1
Data: 20/04/2013

As informações contidas neste documento estão sujeitas a alterações sem prévio aviso.
Este manual foi elaborado por:
DIGICON S.A.

Setor de Engenharia de
Desenvolvimento de Sistemas

1. Histórico de revisões do manual

Versão	Data	Descrição
1.0	21/03/2012	Criação deste manual.
1.1	26/06/2012	1. Estruturação do manual e entrega da versão aos primeiros desenvovedores e 2. Documentação dos procedimentos da utilização do JNI (jni4net) com DFS.
1.2	17/07/2012	1. Ajustado parâmetros e exemplo LoadList. 2. Ajustado parâmetros e exemplo DeleteList. 3. Alterado a disposição dos quadros de históricos de revisões do manual e atualizações do DFS para o início do manual, permitindo fácil acesso as alterações. 4. Na seção corrigido nome do comando para ConfigFirmware. 5. Ainda na seção 7.8 foram revistos todos os exemplos de respostas melhorando a disposição e tornando mais claro. 6. Substituição do nome do manual de |"Manual Acesso" para "Manual SDK Acesso", pois será parte integrante do SDK; 7. Incluído do SDK um exemplo de projeto em C#. Incluído no SDK o documento de referência dos requisitos;
1.3	27/07/2012	1. Incluido exemplo e parâmetros Eventos. 2. Ajustado parâmetros e exemplo BackupAccess. 3. Ajustado parâmetros e exemplo de BackupAlarm. 4. Ajustado exemplo de carga de lista de acessos LoadList. 5. Ajustado parâmetros ListStatus.
1.4	24/08/2012	1. Alterado o método ExitAction para configuração de MCANet com catraca. 2. Adicionado métodos do retorno do comando CheckBiolist. 3. Adicionado exemplo de retorno do evento de template registrada. 4. Corrigido no item 7.8.1 "StandardOn" estado padrão para NF/NA. 5. Detalhado o item WaitTime . 6. Adicionado referêcia a inclusão do firmware ao SDK.
1.5	05/10/2012	1. Inserido tabela Histórico de atualizações do Firmware. 2. Alterado descrição do campo ExeActuator para coleta de biometria com autorização de cartão mestre. 3. Inserido descrição de Requisição de Biometria item . 4. Requisição de Dados de Pessoa Relacionada a Cartão item 5. Requisição de Dados de Pessoa Relacionada a Identificador item
1.6	23/11/2012	1. Inserido retorno assíncrono nos comandos. 2. Inserido descrição da estrura das FAIXAS ALEATÓRIAS no comando ConfigFirmware. 3. Inserido resposta "Resposta Mensagem", novo item. 4. Inserido os tipos de eventos possíveis na seção de Eventos.

2.1	19/02/2013	1. Alterado a codificação do manual, acompanhando definição dos demais manuais de protocolos da divisão de acesso, conforme capítulo 5.1. 2. Mudança de tipo da variável "Password" dos objetos de validação, era do tipo array de bytes, agora é uma String de 6 caracteres(números). 3. Atualizado comando UpdateBio. 4. Alterados definições de propriedades dos comandos, retornos e eventos de maneira que fique mais clara a interpretação dos mesmos.
3.1	20/04/2013	1. Alterado o nome do manual, inserindo em seu título a designação “Ponto”. 2. Incluído tópicos relacionados aos dispositivos do tipo RELOGIO (DIGIREP e DIGICP). 3. Estruturação das sessões do manual segregando: Comandos do servidor DFS, Comandos comuns aos dispositivos, Comandos especialistas de Ponto e Acesso.

2. Histórico de atualizações do DFS

Versão	Data	Descrição
1.0.0	26/06/2012	1. Versão original compatível com firmware 1.0.0, contemplando funcionalidade para portaria 59. 2. Incorporado ao pacote a integração com Java através das bibliotecas jni4net.
1.1.0	10/07/2012	1. Inserido tratamento de erro no socket handler, estava explodindo no DFS uma exception do .NET (system.io); 2. Corrigido erro no retorno do comando CheckBioList; 3. Corrigido erro no retorno do comando AutoProcessResp; 4. Corrigido na estrutura de retorno de requisição de cartões e, adicionado o bloco das pendências. 5. Correção na classe de retorno do ID de Pessoa, pois a mesma estava como private, o correto é pública.
1.2.0	25/07/2012	1. Corrigido erro no retorno do comando BackupAccess. 2. Corrigido erro no retorno do comando BackupAlarm. 3. Corrigido recebimento de eventos. 4. Implementado enumerators em jdfs. 5. Corrigido erro na coleta dos templates. 6. Incluido campoEnclosureCode (Código do recinto). 7. Alterado o campo SmartCardValidation para aceitar validação do tipo portaria 59, 2 = Portaria 59.
1.2.1	23/08/2012	1. Corrigido retorno do evento de Template Registrado. 2. Corrigido/implemetado métodos no retorno do comando CheckBiolist. 3. Corrigido erro na coleta de templates. 4. Alterado o método ExitAction para configuração de MCANet com catraca. 5. Corrigido estrutura para receber eventos, foi implementada a estrutura padrão.
1.3.0	25/09/2012	1. Versão compatível com firmware 1.2.0. 2. Alterado o campo ExeActuator da estrutura das Funções.Alteração verifica ou não o uso do cartão mestre para cadastro de templates. 3. Permite conexão de firmwares de versões anteriores garantindo somente conexão para atualização via comando UpdateFirmware. 4. Permite atualizar os fontes utilizados no display do MCANet através de envio do pacote . 5. Correção de resposta de de Requisição de Biometria item . 6. Correção de resposta de Dados de Pessoa Relacionada a Cartão . 7. Correção de resposta de Dados de Pessoa Relacionada a Identificador .
2.0.0	23/11/2012	1. Não mais utilizado a propriedade "SpecificParameters " em EVENTOS. 2. Alterado o número de leitoras de 16 para 22. 3. Suporte para leitoras MRA. 4. Suporte ao Java, implementado enumerators no package Util do arquivo jdfs-10.x.x.x .Exemplo de uso contido no projeto exemplo java contido no SDK. 5. Implementado suporte a operações sobre listas - OperationType- no comando LoadList, possibilita as opções de inserir novos registros, sobrepor registros existentes e apagar

		registros.Obs:Válido apenas para listas de Bloqueio e Desbloqueio, para as demais listas, setar OperationType com valor "0". 6. Corrigido retorno de Eventos. 7. Corrigido retorno dos Processos Automáticos(AutoProcessResp).
2.1.0	25/01/2013	1. Mudança de tipo da variável "Password" dos objetos de validação, era do tipo array de bytes, agora é uma String de 6 caracteres(números). 2. Inserção de criptografia a propriedade password. 3. Corrigido comando UpdateBio. 4. Mudança na variável ReaderType de inteiro para String, representando uma mascara.
3.0.0	12/04/2013	1. Corrigido CommStatus tornando possível operar a controladora em modo on/off line. 2. Implementado função Session no primeiro nível do framework para obter informações sobre os equipamentos logados. 3. Implementado comandos para dispositivos do tipo Ponto, para atender as portarias 1.510 e 373: ManagerCompany, RequestLog, RequestBackupLog e RequestPersonList. 4. Ajustado comandos do tipo Acesso para atender os novos dispositivos do tipo Ponto, ficando comuns ao framework: UpdateDateTime, DeviceStatus, LoadList e DeleteList. 5. Inplementado comando DeviceConfig para atualização do firmware via arquivo ou stream.
3.1.0	20/04/2013	1. Implementado funcionalidade para configurar leitora RFID para que seja retornado em sua composição todo serial lido para Aba Track (tipo = 3), preservado: Facility Code + Serial, Wiegand (tipo = 1) e Apenas serial, Wiegand (tipo = 2).

3. Histórico de atualizações do Firmware Acesso

Versão	Data	Descrição
1.0.0	17/07/2012	1. Primeira versão.
1.1.0	24/08/2012	1. Controle da catraca MCAFIT com MCANet através da placa controladora. 2. Implementado o monitoramento de braço forçado, com indicações luminosas correspondentes.
1.2.0	08/10/2012	1. Utilização de função da coleta de biometria com autorização de cartão mestre; 2. Aumento dos fontes utilizados no display da MCANET; 3. Inclusão da nova libmcanet 1.4.1; 4. Corrigido bug quando o sensor Sagem não tinha biometria cadastrada, o firmware não fazia a verificação se estava vazio, então quando o firmware precisa fazer alguma operação o normal era cancelar o handler, neste caso como não estava inicializado ocorria erro. 5. Corrigido atualização do display que ficava escrevendo todo tempo, ao passo que é necessário apenas quando é solicitado e depois a cada minuto que é o tempo do relógio 6. Corrigido bug quando iniciava a coleta da biometria e o processo era cancelado a rotina já tinha excluído o template anterior, ficando o usuário sem template 7. Incluída mensagem no display em caso de erro na leitura do código de barras (apenas MCA), antes aionava dois beeps. 8. Corrigido bug da tecla "seta cima" trocada pela tecla MENU na função de configuração de rede. 9. Ajustes na função de apresentação do cursor no display do MCANet para funcionamento com fontes ttf. 10. Aumentado para 2 segundos o tempo de cada mensagem de falha do cadastro biométrico. 11. No cadastro biométrico, aguarda o usuário retirar o cartão da leitora para efetuar a validação por cartão mestre. 12. Retirado acentos em mensagens enviadas ao console.
2.0.0	23/11/2012	1. Corrigido problema de efetuar várias validações quando o cartão mifare está no limiar da distância de leitura da antena; 2. Modificada a mensagem de versão de homologação no display; 3. Inclusão dos módulos remotos de acesso no looping das leitoras, posição 17 a 22; 4. Atualização da biblioteca estática para MCANet (libmcanet_1.4.1.a); 5. Agregada ao comando de atualização de firmware (comando 13), a atualização dos dispositivos MRAs, quando presentes; 6. Agregado no retorno do comando de status de dispositivo (DeviceStatus), o estado de bloqueio ou não dos dispositivos MRAs quando presentes; 7. Agregados ao comando de atualização de data e hora), MRA 8. Atualização do Horário de Verão , os dispositivos MRAs quando presentes; 9. Agregados ao comando de Bloqueio/Desbloqueio os dispositivos MRAs quando presentes;.

		10. Agregados ao comando de Ativar/Desativar Emergência), os dispositivos MRAs quando presentes; 11. Incluídas configurações de monitoramento de entradas no comando de configuração do firmware, na estrutura de "Entradas X 4" (utilizando apenas as duas primeiras); 12. Incluídas configurações de acionamento de relé em configurações de leitoras, na estrutura para "Ações de Acesso Válido", configurações da "Urna"; 13. Incluídas configurações de acionamento de transistor em configurações de leitora, na estrutura para "Ações de Acesso Válido", configurações do "Solenóide(1)"; 14. Corrigido falha onde apenas quando configurada na tecla 1, a função de cadastro biométrico mediante apresentação do cartão mestre operava; 15. Corrigido exibição intermitente da mensagem de retirar o cartão, quando a entrada era feita pelo teclado no cadastro biométrico; 16. Inclusão de método auxiliar "GetFunctionUseMasterCard" e melhor nominação a algumas variáveis; 17. Portaria 59 (Liberação de acesso Off sem antes solicitar validação da Biometria); 18. Implementação portaria 65 e 31 (Novas Normas para Recinto); 19. Corrigido cadastro de biometria Handkey por teclado (Cadastro de biometria Handkey pelo teclado do dispositivo não funciona).
2.1.0	28/01/2013	1. Exclusão do cartão da lista de acesso (tipo 3) quando a pessoa passar pela catraca, caso o campo "CheckList" da leitora esteja em 2; 2. Manipulação dinâmica das listas de feriado (tipo 2) e senhas (tipo 8); 3. Mudança na estrutura de diretórios onde, para atualizar o firmware da MCA, é preciso enviar via comando UpdateFirmware o arquivo firmware-digicon-ppc.v2.1.tar.gz e após este, enviar o arquivo firmware-digicon.v2.1.tar.gz .Para a MCANet, basta enviar o arquivo firmware-digicon.v2.1.tar.gz ;
3.0.0	08/04/2013	1. Incluídos no fimware novas bibliotecas para MRA. 2. Inclusão do campo mensagem para todas as respostas de requisições de acesso. 3. Implementado no comando BackupAlarme para retornar os alarmes ocorridos nas MRA's. 4. Ajustado o comando DeviceStatus para informar o estado da leitora 13 (teclado). 5. Inserido a leitora 13 aos comandos BlockDevice e UnblockDevice, pode-se bloquear ou desbloquear a leitora teclado pelo comando. 6. Implementação de atualização do firmware da MRA através da MCANet, através do comando UpdateFirmware. Obs: Foi inserido o pacote de atualização da MRA junto ao pacote de atualização da MCANet. 7. Correção na resposta do campo LastUpdateFirmware do comando DeviceStatus.
3.1.0	20/04/2013	1. Implementado rotina para tratar nova composição de leitora RFID para que seja retornado em sua composição todo serial lido para Aba Track (tipo = 3), preservado: Facility Code + Serial, Wiegand (tipo = 1) e Apenas serial, Wiegand (tipo = 2).

Sumário

4. Objetivos

O objetivo deste documento é dar uma visão clara e concisa sobre o Digicon Framework Service (DFS), de forma a possibilitar às equipes de desenvolvimento de aplicações conhecimento de como funciona essa ferramenta.

Daremos ênfase na arquitetura, protocolo de comunicação, funcionamento (funções, alertas e eventos), suas qualidades e especialidades.

5. Definições Gerais

Definições gerais para interpretação de padrões de nomeclaturas de nomes de arquivos.

5.1. Codificação de Revisões

Visando melhor entendimento, compatibilidade, distribuição e facilidade nas manutenções, estabelecemos a codificação deste manual seguindo "**P.R**", onde:

P = Protocolo, o número do protocolo do firmware que será incrementado toda vez que ocorrer uma alteração no datagrama do firmware.

R = Revisão do documento, será incrementado para toda alteração dentro da codificação do Protocolo (P), ou qualquer alteração na estrutura do manual, seja qual for, iniciando sempre do algarismo 1, acompanhará todas implementações de novas funcionalidades, sem quebra de compatibilidades e correções refletindo diretamente nas mudanças de Versões e Build do firmware que deverá ser anotada no quadro revisões deste documento.

6. Pré Requisitos

A preparação do ambiente de desenvolvimento é de fundamental importância para que o DFS seja instalado e executado corretamente. Assim, é necessário que antes de proceder a instalação do DFS em seu computador e efetuada a configuração seja preparada a infra-estrutura básica de softwares conforme orientações a seguir.

6.1. Ambiente Windows

Para que haja compatibilidade entre o DFS e o ambiente de desenvolvimento de aplicações é necessário que a infra-estrutura esteja montada com os seguintes componentes:

Sistema operacional: MS Windows 7 Service Pack 2 ou versão superior,32 ou 64 bits, sendo necessário o uso de compilação adequada para cada uma das versões escolhidas;
MS Visual Studio 2010 ;
Framework: MS Dot Net Framework 4.0 .

Observação: Caso não possua licença do MS Visual Studio uma versão simplificada (banner) poderá ser obtida no seguinte endereço:http://www.microsoft.com .

6.2. Ambiente Linux

Neste primeiro momento não foi prevista a utilização do framework DFS no sistema operacional Linux. Isto posta, esta documentação não contém orientações de instalação e configuração para distribuições do Linux.

7. Preparação do ambiente de desenvolvimento em ambiente Windows

Quando houver necessidade de prestar manutenção corretiva ou desenvolvimento de novas funcionalidades para o DFS é necessário que a máquina do desenvolvedor contenha os softwares abaixo:

- Integrated Development Environment – IDE, ambiente de desenvolvimento "Visual Studio 2011" que possui todo o conjunto de ferramentas necessárias, bem como do framework DOT NET na versão 4.0

- O sistema operacional MS Windows 7 instalado e configurado .

8. Arquitetura do framework DFS

Com a finalidade de uniformizar as informações sobre a preparação do ambiente de desenvolvimento e sua infra-estrutura básica apresentaremos nesta documentação alguns comandos e informações são divergentes entre as diversas linguagens de programação, que poderão inviabilizar o funcionamento do DFS. Assim, o conhecimento das linguagens envolvidas é de extrema importância para a conclusão do projeto.

A título de exemplo apresentamos algumas divergências:

a. Utilizado o JAVA é possível se utilizar de variáveis em *byte*, contudo, se essa mesma informação for apresentada em linguagem "C" deverá ser apresentada como variável o "char";

b. À ordem significativa das informações no bloco de dados, que variam quando se utiliza JAVA (BIGENDIAN) e necessita se comunicar com os *firmwares* dos dispositivos que são desenvolvidos em "C Ansi" ou "C^{++}" (LITTLEENDIAN), pois deverá ser convertido o formato antes de ser enviado do firmware para o DFS ou vice e verso.

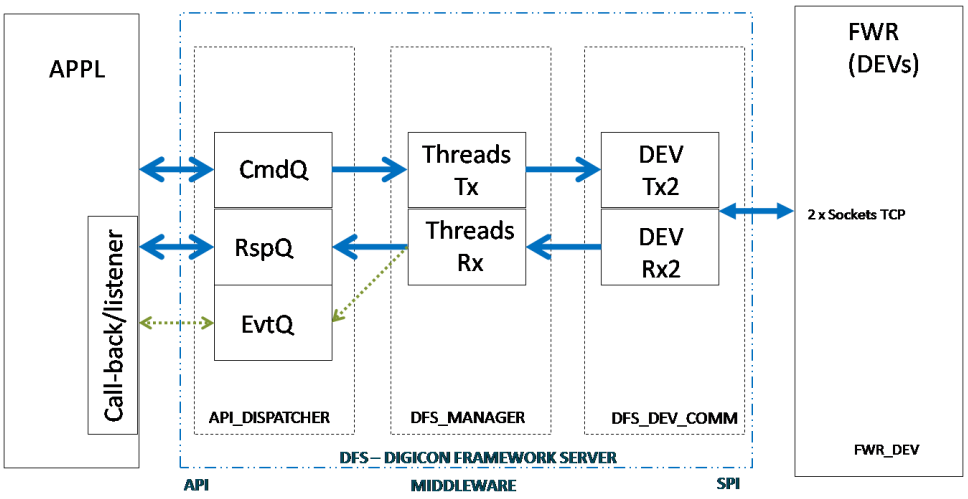

Figura 01 - Visão Geral da arquitetura da solução de acesso com DFS

8.1. Funcionamento do Framework DFS

A aplicação (APP) pode utilizar o DFS de 2 (duas) maneiras distintas:

a. Ativa (Recomendado) – neste caso a aplicação disponibiliza uma funçãode callback/listener antes de executar o Start.Assim sendo após a aplicação executar o comando Start, o DFS_MANAGER irá chamar a função de callback/listener da aplicação sempre que tiver um evento ou respostas de comando que fora enviado ao DFS ou firmware previamente. Desta maneira as filas de respostas raramente ficarão com dados retidos e a aplicação não precisa consultar as filas de respostas "fazer polling" para saber se existe algum evento ou resposta a processar.

b. Passiva – A aplicação executa o Start sem disponibiliza funções de callback/listener. Neste caso a aplicação deverá consultar frequentemente as filas de respostas "fazer polling". Este processo se mal codificado pode causar excessivo consumo de processamento se consultadas com muita freqüência sem liberar CPU, ou mesmo enchimento de filas com risco de perca de dados ou sincronismo com o firmware se a aplicação demorar muito tempo para remover as informações retidas nas filas. Por isso é extremamente recomendável o uso de call-back/listener na aplicação.

8.2. Inicialização

Antes de a aplicação chamar a função Start, é recomendado que a mesma passe para o DFS o ponteiro da função de callback/listener/Delegate para que o DFS possa notificar a aplicação sempre que tiver algum dado a ser enviado como: Evento ou Resposta de comando. "Modo Ativo". Para inicializar o DFS a aplicação deve chamar a função Start. Quando esta função é chamada pela primeira vez, inicializa o DFS (Digicon Framework Server). Após ter sido inicializado o DFS levanta seu callback/listener/Delegate (Socket TCP) para os equipamentos com firmware Digicon instalados na redrav. O Firmware dos dispositivos Digicon é Ativo e quando offline fica periodicamente tentando se conectar com o callback/listener/Delegate do server cujo endereço IP e porta (Socket TCP) estão pré-configurados, Assim sendo, logo após o Listener dos dispositivos estiver ativo, começará a receber conexões través do DEV_DFS_COMM. Neste momento o DFS verifica a versão do firmware e estando de acordo

estabelece 2 (duas) conexões para cada firmware. Uma para recebimento de eventos e outra para comandos e respostas. A partir deste ponto o DFS manterá contato com todos periféricos, recebendo deles todos os alertas e eventos, inclusive eventos periódicos de keep_alive quando não existirem informações a serem transferida. Isso permite que ambos (DFS e Firmware) saibam que o outro permanece ativo e conectado. Sempre que um firmware Digicon estiver conectado o DFS registra esta informação possibilita o envio de eventos do Firmware para o DFS ou Aplicação, recepção de comandos da Aplicação ou DFS para o Firmware e retorno de respostas (síncronas ou assíncronas) desses comandos.

8.3. Comandos Aplicação (APP) <> Framework (DFS) <> Firmware (FWR)

Quando um Firmware estiver conectado, o DFS retém seu identificador (ID). A Aplicação poderá então enviar comandos para os dispositivos através da função “Execute”. O DFS_Manager captura este comando, analisa e converte o comando bem como seus parâmetros para pacotes de comandos conhecidos pelo firmware. Uma vez feito isso este comando é enviado ao canal de comandos, respeitando o protocolo utilizado pela versão de firmware daquele dispositivo. Se o comando for (Síncrono) a resposta será devolvida imediatamente, mas, se o comando for (Assíncrono) “Grande Maioria” o Firmware reportara que recebeu o comando com sucesso e esta em execução. Caso haja alguma inconsistência como: Comando desconhecido, pacote ou parâmetros inválidos, o firmware reportará imediatamente a falha e dependendo do caso, poderá até reiniciar sua conexão.

8.4. Finalização do DFS

Para finalizar o Servidor DFS basta executar o comando Stop. Todavia, o DFS_Manager irá primeiro aguardar chegada das respostas de todos comandos em execução enviadas ao firmware e esperar que a aplicação tenha sido notificada de todos os eventos e que todas as filas estejam vazias.

9. Protocolo de Comunicação APP <> DFS

9.1. Visão Geral

Este capítulo tem por finalidade apresentar as definições de todas as funções, parâmetros, retornos e eventos, que deverão ser utilizadas no processo de integração com uma aplicação de acesso (API) com o Framework DFS.

O DFS não retém nenhuma informação referente ao negócio, somente as necessárias para o funcionamento e manutenção das sessões com os dispositivos Digicon, ou seja, identificação dos endereços dos periféricos, seus status, mantém a comunicação com os mecanismos conectados (keep-alive), etc.

Será de responsabilidade da aplicação envio de comandos, tratamento de retornos e eventos.

Esclarecemos que o Framework DFS possui função de gerenciamento, em tempo de execução, que permite inicializar e desabilitar logs como também obter informações sobre a sessão.

9.2. Passos Iniciais

Nesta seção serão abordados os passos iniciais para desenvolvimento de aplicações nas linguagens C# e Java.

9.2.1. SDK Acesso e Ponto Digicon

Após descompactar o pacote SDK (Software Development Kit), o desenvolvedor encontrará a seguinte estrutura:

\bin

firmware-digicon.vx.x.x.tar.gz ➔Firmware acesso compatível com DFS atual.
firmware-digicon-mca-base.vx.x.x.tar.gz➔Firmware base para atualizações de equipamentos MCA (PowerPC) com firmware com versões inferiores a 2.1.0 é obrigatório passar por esta versão se precisar atualizar superiores, devido o tamanho do pacote versus espaço físico na memória flash das controladoras MCA.

fontesTTF-digicon.tar.gz➔ Pacote dos fontes utilizados no display do MCANet apenas.
DigiconFrameworkServerTest.exe➔ Aplicação teste compilada com todos os comandos e respostas.
ExpandableGridView.dll➔ Componente da aplicação teste.

\docs

Manual SDK Acesso e Ponto – Desenvolvedor vx.x.pdf ➔ Manual do desenvolvedor contendo instruções de uso e exemplos (este manual) e
Manual SDK Acesso - Referência Requisitos do Projeto vx.x.pdf ➔ Manual contendo explanação dos requisitos do projeto acesso e seus limites.

\lib

Pasta com as bibiotecas de integração, seguintes e significados
dfs-10.x.x.x.dll➔ Biblioteca principal do DFS (Digicon Framework Server).
jni4net.n-x.x.x.x.dll ➔ Biblioteca de integração Java.
jni4net.n.w32.v4-x.x.x.x.dll ➔ Biblioteca de integração Java.
jdfs-10.x.x.x.jar ➔ Biblioteca de abstração principal para integração Java.
cdfs-10.x.x.dll ➔ Biblioteca de integração Java.
jni4net.j-x.x.x.x.jar ➔ Biblioteca de integração Java.
p32dfs-10XXX.dll➔ Biblioteca do DFS.
p64dfs-10XXX.dll➔ Biblioteca do DFS.

\sample

Pasta com exemplos de projetos nas linguagem Java e C#.

9.2.2. C#

Inicialmente, adicionar referência a DLL dfs-10.v.r.b.dll ao projeto.

Segue abaixo a lista das funções primárias exportadas pelo DFS Server para API:

```
public int Start(String localaddr, int port, int online);
```

```
public int Stop();
public int Execute(GenericObject comm);
public MessageReceivedArgs(Message msg, QueueType queueType);
```

Segue abaixo a lista das funções auxiliares exportadas pelo DFS Server para API:

```
public void Ack(int deviceID);
public void Nack(int deviceID);
public Message createMessage(int deviceID, MessageType type, byte[] package);
public Message GetMessage(QueueType queueType);
public void SetMessage(Message message, QueueType queueType);
public void CleanAllMessage() ;
public WriteLogArgs(String deviceID, LogWriterType type, String log);
public void WriteLog(LogWriterType type, String log);
public void WriteLog(LogWriterType type, String log, String id);
public override String ToString();
public void setFile(string file) ;
```

Os primeiros procedimentos a serem adotados para a integração de uma aplicação ao framework DFS deverá ser os seguintes:

```
private DigiconServer DFS = new DigiconServer();
```

Permite que a aplicação tenha acesso a todos os objetos do DFS Server.

Usando delegate para callback:

```
MessageManager.MessageReceivedChanged += new
MessageReceivedHandler(MyListener);

MyListener =Nome do método de escuta da aplicação.

public void MyListener (MessageReceivedArgs args)
{
        // Rotina de tratamento das mensagens vindas do Firmware Digicon.
}
```

Os demais procedimentos serão descritos a seguir:

Para iniciar o DFS, a seguinte sequencia de comandos pode ser aplicada:

```
Int rc;
        rc = DFS.Start("172.16.0.2",3232,2);

        //Este comando necessita receber trêsparâmetros:
        //endereço IP ,número da Porta de Comunicação e o status de comunicação

        if (rc == 0)
        {
                Console.WriteLine("DFS started");
                //Sucesso
        }
```

```
else
{
        Console.WriteLine("DFS Not started");
        Return rc;
         // Falha
}

//Coloque aqui seu código onde se podem executar comandos…
```

Para o envio de comandos para que o DFS execute comandos Execute(), a seguinte sequência de comandos pode ser aplicada:

```
MyCommandObject obj = new MyCommandObject ();
        obj.CommStatus = 2;
        obj.DeviceID = 1;
        obj.SeqCmd = 1;

DFS.Execute(obj);
```

Note-se que MyCommandObject significa qual comando que se deseja executar. As três linhasseguintes setam os parâmetros necessários para executar o referido comando no dispositivo identificado pelo ID .

Para encerrar o DFS, a seguinte sequência de comandos pode ser aplicada:

```
DFS.Stop();
Console.WriteLine("DFS stopped");
```

9.2.3. Java

Inicialmente,adicionar referência aos seguintes arquivos:

/bin/
dfs-10.x.x.x.dll➔ Biblioteca principal do DFS (Digicon Framework Server).
jni4net.n-x.x.x.x.dll ➔ Biblioteca de integração Java.
jni4net.n.w32.v4-x.x.x.x.dll ➔ Biblioteca de integração Java.
jdfs-10.x.x.x.jar ➔ Biblioteca de abstração principal para integração Java.
cdfs-10.x.x.dll ➔ Biblioteca de integração Java.
jni4net.j-x.x.x.x.jar ➔ Biblioteca de integração Java.
p32dfs-10XXX.dll➔Biblioteca do DFS.
p64dfs-10XXX.dll➔ Biblioteca do DFS.

Após as referências serem criadas,carregar a dll:

```
Bridge.init();
Bridge.LoadAndRegisterAssemblyFrom(new File("./cdfs-10.x.x.x.dll"));
```

Abaixo a lista das funções primárias exportadas pelo DFS Server para API:

```
public int Start(String localaddr, int port, int online);
public int Stop();
```

```
public int Execute(GenericObject comm);
public MessageReceivedArgs(Message msg, QueueType queueType);
```

Segue abaixo a lista das funções auxiliares exportadas pelo DFS Server para API:

```
public void Ack(int deviceID);
public void Nack(int deviceID);
public Message createMessage(int deviceID, MessageType type, byte[] package);
public Message GetMessage(QueueType queueType);
public void SetMessage(Message message, QueueType queueType);
public void CleanAllMessage() ;
public WriteLogArgs(String deviceID, LogWriterType type, String log);
public void WriteLog(LogWriterType type, String log);
public void WriteLog(LogWriterType type, String log, String id);
public override String ToString();
public void setFile(string file) ;
```

Os primeiros procedimentos a serem adotados para a integração de uma aplicação ao framework DFS deverá ser os seguintes:

```
private DigiconServer DFS = new DigiconServer();
```

Permite que a aplicação tenha acesso a todos os objetos do DFS Server.

Usando Listener para callback:

```
  MessageManager.addMessageReceivedChanged(
new MessageReceivedHandler()
{
        @Override
publicvoid Invoke(MessageReceivedArgs args)
{
            // Rotina de tratamento das mensagens vindas do Firmware Digicon.
            }
```

Para iniciar o DFS, a seguinte sequencia de comandos pode ser aplicada:

```
Int rc;
            rc = DFS.Start("172.16.0.2",3232,2);

//Este comando necessita receber trêsparâmetros:
            //endereço IP ,número da Porta de Comunicação e o status de comunicação

            if (rc == 0)
            {
                  System.out.println("DFS started");
                  // Sucesso
            }
            else
            {
                  System.out.println("DFS Not started");
                  // Falha
```

```
}

//Coloque aqui seu código onde se podem executar comandos…
```

Para o envio de comandos para que o DFS execute comandos Execute(), a seguinte sequência de comandos pode ser aplicada:

```
MyCommandObject obj = new MyCommandObject ();
        obj.setCommStatus((byte)2);
        obj.setDeviceID(1);
        obj.setSeqCmd(1);

DFS.Execute(obj);
```

Para encerrar o DFS, a seguinte sequência de comandos pode ser aplicada:

```
DFS.Stop();
System.out.println("DFS stopped");
```

9.2.4. Nota

Para que um aplicativo ou DLL escrito em .NET possa chamar uma DLL C/C++ é necessário:

[.NET] -> ---------- -> [C(++)] ... via DllImport Attribute

Para que um aplicativo ou DLL escrito em C/C++ chame DLL em .NET é necessário:

[C/ C(++)] -> [C++/CLR] ->[.NET]… "default" .NET interface + "default" native interface

Exemplo C++ chamando C# DLL :

```
using namespace Your::Namespace::Here;
#using <YourDotNET.dll>
YourManagedClass^ pInstance = gcnew YourManagedClass();
```

where 'YourManagedClass' is defined in the c# project with output assembly 'YourDotNET.dll'.

9.3. Convenções Utilizadas

Objetivos:
Nas declarações das funções dos comandos e retornos estaremos convencionando:

Tipo do Parâmetro, Tamanho (limitado pelo tipo do dado e valor entre colchetes), Nome do Parâmetro e Comentário após duas barras invertidas.

I[10]	port	//Parâmetro numérico inteiro, com limite até 10 dígitos, no exemplo, porta de escuta do Firmware no Servidor DFS ex: 3232 .

N[15]	newDeviceIP	//Array de dígitos separados por ".", específico para configurações de rede, no exemplo IP desejado para o Device ex: "172.16.0.23" .
A [25]	StandardDigitalOut	//Array de dígitos separados por ";", específico para configuração de conjuntos de bytes combinados, no exemplo estado padrão das saídas digitais, 1 byte para cada saída, 1=ativado, 0=desativado, formato "0;0;0;0;0;0;0;0;0;0;0;0;0"
T	Inicio Funções x 10	//Estrutura com aninhamento de várias colunas que podem repetir "n" vezes, teremos um texto "Inicio" e "Fim" delimitando onde começa e onde termina, no exemplo estrutura de "Funções" que são acionadas pelo treclado de "0" a "9", ou seja 10 vezes.
B [1]	Level1	//Dado do tipo byte
D	EventInitDateTime	// Data/Hora Inicial no formato dd/mm/aaaa hh:mm:SS
O	LoadPermissionList	// Objeto com vários valores

9.4. Gerenciamento do Servidor DFS

Os comandos Servidor DFS são para gerenciamento no nível da instancia do DFS (Digicon Framework Server), aplicado a todos tipos de dispositivos.

9.4.1. Abrindo Canal de Escuta - Listener / Callback / Delegate

Descrição:
É o método que a aplicação utiliza para tratar os eventos enviados pelos dispositivos, disponibilizados através do DFS.
O recebimento de mensagens do dispositivo é capturado de forma assíncrona, ou seja, a aplicação deve estabelecer um mecanismo de escuta (Listener em Java, Delegate em C#, etc).

Exemplo:

MessageManager.MessageReceiveChanged+=
new MessageReceivedHandler(MyListener);
MyListener =Nome do método de escuta da aplicação.

// Respostas de eventos do tipo ACESSO

```
public void MyListenerAcesso (MessageReceivedArgs args){
if ((args.Msg.MsgType != MessageType.MSG_ACCESS_REQUEST_CARD) &&
    (args.Msg.MsgType != MessageType.MSG_ACCESS_REQUEST_PERSON) &&
    (args.Msg.MsgType != MessageType.MSG_TEMPLATE_REQUEST) &&
    (args.Msg.MsgType != MessageType.MSG_DATA_REQUEST_CARD) &&
    (args.Msg.MsgType != MessageType.MSG_CONNECTION_REQUEST) &&
    (args.Msg.MsgType != MessageType.MSG_DATA_REQUEST_PERSON) &&
```

```
    (args.Msg.MsgType != MessageType.MSG_SYNC_RETURN) &&
   !(args.Msg is EventConnChangeStatus))
    {
     LogWriter.Instance.WriteLog(LogWriterType.DEBUG, "Delegate - Enviando ACK");
     DFS.Ack(args.Msg.DeviceID);
    }

if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_CARD)
   {
        // Codigo de tratamento de evento de acesso com cartão.
   }
        // Testar outras mensagens aqui...
   }

// Respostas de eventos do tipo PONTO

public void MyListenerRelogio (MessageReceivedArgsClock args)
{
        // Código de tratamento de eventos de relógio
}
```

9.4.2. Iniciando o Servidor – Start()

Descrição:
Esta função inicializa o Servidor DFS e coloca a porta de escuta (TCP) ativa, permitindo que os dispositivos se conectem para que enviem eventos e respostas.

Cada vez que um dispositivo se conecta ao DFS é verificada a versão do firmware e estando de acordo, temos as seguintes particularidades por tipo:

Acesso: estabelece 2 (duas) conexões para cada mecanismo, uma para recebimento de eventos e outra para comandos e respostas, sendo que trabalharão síncrono e asincrono.

Ponto: estabelece 1 (uma) conexão que trabalhará em modo síncrono.

Chame e função MessageReceivedHandler antes da execução deste comando.

Função:
Int Start(String localaddr, int port, int online);

Parâmetros:

S[15]	localaddr	//Endereço IP do Servidor DFS string Exemplo: “172.16.0.2”
I [5]	port	//Porta de escuta do dispositivos no Servidor DFS ex: 3232
I [1]	online	//Estado desejado do Firmware, 1=OffLine, 2=OffLine

Retorno:
0 = Successo / -1 = Falha

Exemplo em C#

```
private DigiconServer DFS = new DigiconServer();
int online = 2;
```

```
if (DFS.Start("172.16.0.2",3232,online) < 0)
   {      Console.WriteLine("Erro");
   }
else
   {
          Console.WriteLine("DFS started");
   }

// A partir deste ponto o DFS esta ativo.
```

9.4.3. Parando o Servidor – Stop()

Descrição:
Esta função termina o servidor DFS e fecha a porta TCP de escuta com os dispositivos.
O DFS irá primeiro aguardar chegada das respostas de todos os comandos em execução enviadas ao firmware e esperar que a aplicação tenha sido notificada de todos os eventos e que todas as filas estejam vazias.

Função:
Int Stop();

Parâmetros:
Nenhum

Retorno:
0 = Successo / -1 = Falha

Exemplo:

```
DFS.Stop();
Console.WriteLine("DFS stopped");
```

9.4.4. Enviando Comandos e Respostas – Execute()

Descrição:
Esta função envia comandos e respostas ao firmware através do DFS.

Função:
Int Execute(Obj);

Descrição dos Parâmetros comuns a todos comandos:

```
obj.CommStatus      // Estado da comunicação,1=Offline, 2=Online
obj.DeviceID        // ID do dispositivo para onde deverá ser enviado o comando
obj.SeqCmd          // Sequencial do comando
```

Exemplo:

Enviando Comandos pela Aplicativo através da API do DFS:

O Aplicativo deve em primeiro lugar criar um objeto local com o nome do objeto exportado pelo DFS, neste caso é o UpdateDateTime(), uma vez criado o objeto local, pode atribuir valores a seus atributos. Os parâmetros comuns são:

CommStatus: Esta flag é enviada pela APP atrabés do DFS, ou seja, quem comanda o "modus operandis" do firmware é a APP, se for enviado 1 o firmware entrará em off-line de fato e passará a validar as regras através das listas anteriormente carregadas e colocará um pontinho "." na tela o de baixo, se o dispositivo apresentar dois pontinhos ":" na linha baixo não tem comunicação com DFS provável handshake errado. Sempre que a APP quiser que o dispositivo opere em modo on-line deve ser enviado 2.

DeviceID: Identificador do dispositivo que se quer acessar, este deve ser único na rede, este id é mantido pelo DFS para identificação dos comandos, retornos e eventos gerados pelo firmware de cada dispositivo e

SeqCmd: É um valor seqüencial controlado pela aplicação, é um número de 32 bits, ou seja é possível armazenar até 10 dígitos, com valor máximo de 4294967295, ou seja "FFFFFFFF". Aconselhamos que a APP controle por comando, pois este atributo será devolvido ao listener na chegada do evento de resposta de comandos assíncronos ou síncronos, sendo assim a APP poderá diferenciar caso haja comandos, ou seja, a cada comando igual SeqCmd=SeqCmd+1.

```
UdateDateTime obj = new UpdateDateTime();
        obj.CommStatus = 2;                    //2=Online
        obj.DeviceID = 1;                      //1=ID Dispositivo desejado
        obj.SeqCmd = 123;                      //nnn=Sequencial
        obj.DateTime = "18/05/2012 11:19:00";  //Data/Hora

DFS.Execute(obj);                              //Executando comando
```

Recebendo respostas ou eventos através do Listener da Aplicação:
Quando o DFS recebe uma resposta ou evento do dispositivo a mesma é previamente formatada para o layout esperado pela aplicação e o DFS chama o Listener colocando esta resposta em uma fila de tratamento. A Aplicação que tem controle sobre os comandos enviados através do SeqCmd verificará se o comando enviado foi processado com sucesso ou não e se tiver dados a receber estarão na lista de argumentos do retorno.

9.4.5. Solicitando Informações da Sessão – Session( )

Descrição:
Esta função retorna informações dos dispositivos logados.

Função:

```
DeviceList[] Session ();                  //retorna lista de todos os dispositivos logados.
DeviceList Session (int idDispositivo);   //retorna dados apenas do id solicitado.
```

Dados da Resposta do comando:

I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado os comandos
I[1]	DevType	// Tipo do dispositivo, 9 = ACESSO, 2 = DIGIREP, 5= DIGICP
A[15]	IpDevice	// Ip do dispositivo
I[]	FirmwareVersion	// Versão do firmware, ex: 9100, 2180, 5190

```
I[]     Protocol            // Versão do protocolo
D       Login               // Data do último login
```

Exemplo C#:

```
Session[] sessions = DFS.Session();

String stat = "";

foreach (Session session in sessions)
        {

            stat += "\nDev Id = " + session.DeviceID.ToString();
            stat += "\nDev Type = " + session.DevType;
            stat += "\nDev Serial = " + session.Serial.ToString();
            stat += "\nDev Ip = " + session.IpDevice;
            stat += "\nDev Fwr vs = " + session.FirmwareVersion.ToString();
            stat += "\nDev Protocol = " + session.Protocol;
            stat += "\nDev Date Login = " + session.Login.ToString();
            stat += "\nDev Status = " + session.Status.ToString() + "\n";
                }

Console.WriteLine(stat);
```

9.5. Comandos Comuns para Gerenciamento de Dispositivos de Ponto e Acesso

Entende-se sob comando as ações executadas pela APP que através do DFS são validadas e enviadas ao FWR, cujas respostas podem ser Síncronas e Assíncronas.
Nesta sessão serão listados os comandos comuns aos dipositivos de Ponto e Acesso.
Observar que dependendo do tipo do dispositivo, existirão atributos especializados.

9.5.1. Configuração de Dispositivos – DeviceConfig()

Descrição:
Esta função executa comando de configuração de um dispositivo alvo, possibilitando que as configurações seja carregadas a partir de arquivos gerados pelo Configurador Digicon, mesmo que este arquivo seja armazenando em banco de dados da APP em campo do tipo BLOB ou similar, ou seja, a APP enviará um stream de bytes para este método.
Caso o dispositivo seja do tipo Ponto apenas será possível configurar o tempo de keep alive e o modo de validação biométrica, isto porque as regras legais que são aplicadas a estes dispositivos exigem que o equipamentos saiam configurados de fábrica.
Quando o dipositivo for do tipo Acesso, o DFS internamente utilizará o comando ConfigFirmware para o envio, ou seja, o desenvolvedor tem aqui um método de alto nível para configurar este tipo de dispositivo que por vasta aplicação de uso, sua configuração se torna complexa.

Observação: Na versão 3.1.0 o arquivo de configuração será produzido pelo DFSTools da mesma versão, não serão aceitos arquivos de versões inferiores.

Função:

Int Execute(DeviceConfig);

Parâmetros:

Comuns

I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando

Acesso

I[1]	CommStatus	// Estado da comunicação 1=Offline, 2=Online
I[9]	SeqCmd	// Sequencial do comando
M	Configuration	// Método que recebe : String com o caminho (path) e arquivo de configuração ou byte array com o arquivo de configuração em formato de array de bytes.

Relógio

I [9]	KeepAlive	// Tempo em milesegundos entre pacotes

I[1]	BiometricValidation	// Tipo de validação biométrica
		1 = 1:1, utiliza o id físico do cartão para localizar a pessoa e em segundo passo localiza o template armazenado no mapa do cartão Mifare, caso não exista efetua a marcação de ponto, pois a primeira validação com cartão foi validada.
		2 = 1:N, pesquisa a biometria no banco de dados do módulo Biométrico, se localizado procura na lista de empregados para pegar PIS e Nome.
		3 = 1:1, utiliza o id físico do cartão para localizar a pessoa e apartir do resultado, se existir biometria solicita ao módulo biométrico efetuar a pesquisa no no banco de dados do módulo, caso não localize efetua a marcação de ponto, pois a primeira validação com cartão foi validada.

Recepção de Resposta Assincrona para dispositivos do tipo Relógio:

Vide resposta do comando **ConfigFirmware**.

Recepção de Resposta Assincrona para dispositivos do tipo Relógio:

```
if (args.Msg is DeviceConfigResp )
        {
        DFS.Ack(args.Msg.DeviceID);
        }
```

9.5.2. Status de Dispositivos – DeviceStatus()

Descrição:
Esta função obtem o status de um determinado dispositivo.

Função:
Int Execute(DeviceStatus);

Parâmetros:

Comuns

I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
O	DeviceStatus	// Device Status

Acesso

I [1]	CommStatus	// Estado da comunicação Online=2 Offline=1
I [9]	SeqCmd	// SeqCmd=128 Comando enviado

Retorno do envio de Comandos:

0 = Comando Enviado com Sucesso ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:

Variáveis comuns

I[9]	DeviceID	// ID do dispositivo
N[15]	DeviceIP	// Endereço IP do Dispostivo
I[7]	FirmwareVersion	// Versão do Firmware

Variáveis para objeto Acesso

D	LastFirmwareUpdate	// Data/Hora Ultima atualização do Firmware
D	LastConfUpdate	// Data/Hora Ultima atualização da Configuração
I [1]	Emergency	// Em Emergencia, 0=não, 1=sim
I [3]	GMT	// GMT (-3) = -180
D	IniSummerTime	// Data/Hora Inicio Horário de Verão
D	EndSummerTime	// Data/Hora Final Horário de Verão

Variáveis para objeto Relógio

D	DateFile	// Momento da geração deste status
D	FwrVersionDate	// Data da versão do firmware
L	MrpNumber	// Número da MRP
S[ ]	MrpRevision	// Revisão da MRP
L	MrpAddressAbsolute	// Endereço absoluto da MRP
L	MrpAddressCurrent	// Endereço corrente da MRP
S[ ]	MrpStatus	// Status da MRP
b	StatusBioReaderSagem	// Status do leitor biométrico Sagem
b	StatusBioReaderSuprema	// Status do leitor biométrico Suprema
b	StatusMifareReader	// Status do leitor Mifare
b	StatusBarReader	// Status do leitor de Código de Barras
b	StatusRfidReader	// Status do leitor RFID
I[ ]	CountUsers	// Número de colaboradores cadastrados
I[ ]	CountRfidCard	// Número de cartões RFID cadastrados
I[ ]	CountBarCard	// Número de cartões de Código de Barras cadastrados
I[ ]	CountMifareCard	// Número de cartões Mifare cadastrados
I[ ]	CountCards	// Número de cartões geral cadastrados (DIGIREP vs1.7.5 ou DIGICP vs1.8.0)
I[ ]	CountBioTemplate	// Número de templates biométricos cadastrados
I[ ]	FlashMaxLenght	// Tamanho máximo da Flash
I[ ]	FlashUsed	// Quantidade utilizada da Flash
I[ ]	RamMaxLenght	// Tamanho máximo da RAM
I[ ]	RamUsed	// Quantidade utilizada da RAM
I[ ]	MrpMaxLenght	// Tamanho máximo da MRP
I[ ]	MrpUsed	// Quantidade utilizada da MRP
I[ ]	RegType2	// Quantidade de registros do tipo 2 (inclusão e alteração de empresa)

I[ ]	RegType3	//Quantidade de registros do tipo 3 (marcação de ponto)
I[ ]	RegType4	//Quantidade de registros do tipo 4 (alteração de data e hora)
I[ ]	RegType5	//Quantidade de registros do tipo 5 (inclusão, alteração, exclusão e alterações de cartões e biometrias do colaborador)
S[ ]	PrinterFwrVersion	// Versão do firmware da impressora
S[ ]	PrinterLevel	// Nível da bobina de papel
I[ ]	CountPrintedTickets	// Número de comprovantes de marcação de ponto impressos
I[ ]	CountPrintedReport	// Número de relatórios 24hs impressos
S[ ]	NetIpserver	// Ip do servidor configurado o DigiREP
S[ ]	NetMask	// Máscara de rede configurada no DigiREP
S[ ]	NetGateway	// Gateway configurado no DigiREP
I[ ]	Netport	// Porta de comunicação configurada no DigiREP
I[ ]	NetCountTimeConServer	// Tempo entre tentativas de conexões com o servido
I[ ]	NetTimeoutMessage	// Timeout de mensagens

Exemplo C#:
Envio do Comando:

```
DeviceStatus obj = new DeviceStatus ();
                obj.CommStatus = 2;
                obj.DeviceID = 1;
                obj.SeqCmd = 128;
if (DFS.Execute(obj) < 0){ /*erro*/
}else{
Console.WriteLine("Comando enviado com sucesso – Aguarde retorno!");
}
```

Recepção de Resposta Assincrona para dispositivos do tipo Acesso:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
  {
  String BuffMsg = "";
   if (args.Msg is DeviceStatusResp)
    {
     DFS.Ack(args.Msg.DeviceID);
BuffMsg =
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID      " + ((DeviceStatusResp)args.Msg).DeviceId.ToString() +
"\nSequencia CMD " + ((DeviceStatusResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((DeviceStatusResp)args.Msg).ReturnCode.ToString() +
"\nData Ger CMD " +
((DeviceStatusResp)args.Msg).CmdGenerationTimestamp.ToString() +
"\nData Exe CMD " + ((DeviceStatusResp)args.Msg).CmdExecutionTimestamp.ToString()
+
"\nTam dos Dados " + ((DeviceStatusResp p)args.Msg).PayloadSize.ToString()+
\nVersion          : " +
((DeviceStatusResp)args.Msg).FirmwareVersion[0] +
((DeviceStatusResp)args.Msg).FirmwareVersion[1] +
((DeviceStatusResp)args.Msg).FirmwareVersion[2] +
"\nLastFirmwareUpdate : " +
((DeviceStatusResp)args.Msg).LastFirmwareUpdate.ToString() +
 "\nGmt    : " + ((DeviceStatusResp)args.Msg).Gmt.ToString() +
"\nLastConfUpdate : " + ((DeviceStatusResp)args.Msg).LastConfUpdate.ToString() +
```

```
"\nInitSummerTime : " + ((DeviceStatusResp)args.Msg).InitSummerTime.ToString()+
 "\nEndSummerTime : " + ((DeviceStatusResp)args.Msg).EndSummerTime.ToString()+
 "\nDeviceIP   : " +
((DeviceStatusResp)args.Msg).DeviceIP[0] +"."  +
((DeviceStatusResp)args.Msg).DeviceIP[1] + "." +
((DeviceStatusResp)args.Msg).DeviceIP[2] + "." +
((DeviceStatusResp)args.Msg).DeviceIP[3] +
"\nEmergencia         : " + ((DeviceStatusResp)args.Msg).Emergency +
"\nDevice Id          : " + ((DeviceStatusResp)args.Msg).DeviceID.ToString()+
"\nReaderBlock : " + ((DeviceStatusResp)args.Msg).ReaderBlock.ToString()
;
      }
Console.WriteLine()
}
```

Recepção de Resposta Assincrona para dispositivos do tio Relógio:

```
if (args.Msg is RequestDeviceInfoResp)
            {
     String msg"\nRetorno do comando RequestDeviceInfo " +
 "\nDeviceID " + ((RequestDeviceInfoResp)args.Msg).DeviceID.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).DateFile.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).FwrVersionDate
"\n " + ((RequestDeviceInfoResp)args.Msg).FwrVersionName
"\n " + ((RequestDeviceInfoResp)args.Msg).MrpNumber.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).MrpRevision
"\n " + ((RequestDeviceInfoResp)args.Msg).MrpAddressAbsolute.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).MrpAddressCurrent.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).MrpStatus
"\n " + ((RequestDeviceInfoResp)args.Msg).StatusBioReaderSagem.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).StatusBioReaderSuprema.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).StatusMifareReader.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).StatusBarReader.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).StatusRfidReader.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).CountUsers.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).CountRfidCard.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).CountBarCard.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).CountMifareCard.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).CountCards.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).CountBioTemplate.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).FlashMaxLenght.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).FlashUsed.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).FlasFree.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).RamMaxLenght.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).RamUsed.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).MrpMaxLenght.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).MrpUsed.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).RegType2.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).RegType3.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).RegType4.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).RegType5.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).PrinterFwrVersion.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).PrinterLevel.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).CountPrintedTickets.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).CountPrintedReport.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).NetIp
"\n " + ((RequestDeviceInfoResp)args.Msg).NetIpserver
"\n " + ((RequestDeviceInfoResp)args.Msg).NetMask
"\n " + ((RequestDeviceInfoResp)args.Msg).NetGateway
"\n " + ((RequestDeviceInfoResp)args.Msg).Netport.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).NetCountTimeConServer.ToString() +
"\n " + ((RequestDeviceInfoResp)args.Msg).NetTimeoutMessage.ToString();
```

```
                DFS.Ack(args.Msg.DeviceID);
Console.WriteLine(msg );
            }
```

9.5.3. Ajustar Data e Hora do Dispositivo – UpdateDateTime()

Descrição:
Esta função atualiza data e hora em um determinado dispositivo.

Função:
Int Execute(UpdateDateTime);

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando
D	DateTime	// Data e hora no formato dd/mm/aaaa hh:mm:ss

Retorno do envio de Comandos:
0 = Comando Enviado com Sucesso ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:
Não há dados de retorno

Exemplo C#:
Envio de Comando:

```
UpdateDateTime obj  = new UpdateDateTime();
                obj.CommStatus = 2;
                obj.DeviceID = 1;
                obj.SeqCmd = 128;
                obj. DateTime = “17/05/2012 21:30:00”;
if (Execute (obj) < 0){
                Console.WriteLine(“Erro");
}else{
                Console.WriteLine(“Comando enviado com sucesso!”);
}
```

Recepção de Resposta Assincrona:

Resposta caso device tipo Acesso

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
      {
      String BuffMsg = "";
      if (args.Msg is UpdateDateTimeResp)
       {
        DFS.Ack(args.Msg.DeviceID);
   BuffMsg =
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID" + ((UpdateDateTimeResp)args.Msg).DeviceId.ToString()+
"\nSequencia CMD" + ((UpdateDateTimeResp )args.Msg).CmdSeq.ToString()+
"\nCodigo de ret " + ((UpdateDateTimeResp)args.Msg).ReturnCode.ToString() +
"\nData Ger CMD " +
((UpdateDateTimeResp)args.Msg).CmdGenerationTimestamp.ToString() +
```

```
 "\nData Exe CMD " +
((UpdateDateTimeResp)args.Msg).CmdExecutionTimestamp.ToString()  +
 "\nTam dos Dados " + ((UpdateDateTimeResp)args.Msg).PayloadSize.ToString();

Console.WriteLine(BuffMsg );
   }
}
```

Resposta caso device tipo Relógio

```
         else if (args.Msg is UpdateClockResp)
            {
BuffMsg +=   "\nRetorno do comando UpdateClock " +
               "\nDeviceID " + ((UpdateClockResp)args.Msg).DeviceID.ToString() +
               "\nStatus " + (ClockResultType)((UpdateClockResp)args.Msg).Status
;


                  DFS.Ack(args.Msg.DeviceID);
Console.WriteLine(BuffMsg );        }
```

9.5.4. Status das Listas – ListStatus()

Descrição:
Esta função carrega a lista de status de determinado dispositivo.

Função:
Int Execute(ListStatus);

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando
B [1]	Type	// Tipo de listas: Lista de permissões = 1 , Lista de Feriados = 2, Lista de Acesso = 3, Lista de Bloqueios = 5 Lista de Refeitorio = 7 Lista de Senhas = 8 Lista de Empregados de Ponto = 9 (**implementação futura**)

Retorno do envio de Comandos:
0 = Comando Enviado com Sucesso ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:

I [5]	obj.QtdRegs	// Quantidade de registros na lista
I [4]	obj.Size	// Tamanho da lista
I [4]	obj.Available	// Quantidade disponível em bytes
D	obj.ListData	// Data e hora da criação da lista (dd/mm/aaaa hh:mm:SS)

Exemplo C#:
Envio do Comando:

```
ListStatus obj = new ListStatus();
                obj.Type = 2;
                obj.CommStatus = 2;
                obj.DeviceID = 1;
                obj.SeqCmd = 123;
                if (DFS.Execute(obj) < 0) {
         } else
        {
                  Console.WriteLine(" Comando enviado com sucesso");
}
```

Recepção de Resposta Assincrona:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
{
 String BuffMsg = "";
if (args.Msg is ListStatusResp)
 {
 DFS.Ack(args.Msg.DeviceID);
BuffMsg =
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID      " + ((ListStatusResp)args.Msg).DeviceId.ToString() +
"\nSequencia CMD " + ((ListStatusResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((ListStatusResp)args.Msg).ReturnCode.ToString() +
"\nData Ger CMD" + ((ListStatusResp)args.Msg).CmdGenerationTimestamp.ToString() +
"\nData Exe CMD" + ((ListStatusResp)args.Msg).CmdExecutionTimestamp.ToString()  +
"\nTam dos Dados " + ((ListStatusResp)args.Msg).PayloadSize.ToString()+
"\nSolicitado = " + ((ListStatusResp)args.Msg).Time.ToString() +
 "\nTipo da Lista = " + ((ListStatusResp)args.Msg).ListType +
"\nQuantidade de Registros = " + ((ListStatusResp)args.Msg).TotReg ;
Console.WriteLine(BuffMsg );
}
}
```

9.5.5. Carregar Listas – LoadList()

Descrição:
Esta função carrega lista de dados para excessão de trabalhar off-line

Função:
Int Execute(ListName);

Parametros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando

B [1]	Type	// Tipo de listas:
		// Lista de permissões = 1 ,
		// Lista de Feriados = 2,
		// Lista de Acesso = 3,
		// Lista de templates biométricos,
		// Lista de Bloqueios = 5
		// Lista de Empregados = 6
		// Lista de Refeitorio = 7
		// Lista de Senhas = 8

		/ /Lista de Empregados de Ponto = 9 (**Apenas Relógios Controle de Ponto**)
O	PermissionList	// 1 = Lista de permissões, apenas utilizada se for trabalhar com smartcard=0 (mapa no cartão)
O	HollidaysList	// 2 = Lista de Feriados, obrigatório para uso de regras de mascaramento e agendamentos de processos bem como firmware quando enviar os eventos indicar se ocoreeu em feriado
O	AccessList	// 3 = Lista de Acesso, análogo a lista de liberação ou lista branca, trabalha na condição "ou" acesso ou bloqueio
O	TemplateList	// 4 = Lista de Templates, lista temporária que é gravada em arquivo até que o firmware envie ao sensor, após isso este arquivo é removido
O	BlockList	// 5 = Lista de Bloqueios, análogo a lista negra Atenção!!! As listas 3 e 5 devem serem manipuladas por opção uma ou outra, caso seja enviado uma lista do tipo 3 e na controladora contiver uma 5, anterior será excluída
O	EmployeeList	// 6 = Lista de Empregados, apenas deve ser utilizada caso for trabalhar com biometria ou digitação de matrícula como forma de acesso
O	SnackTimeList	// 7 = Lista de Faixas Horárias de Refeitório
O	PWdList	// 8 = Lista de Senhas
O	EmployeeListClock	// 9 = Empregados para ponto portaria 1.510 ou 373, gerenciamento unitária
B [1]	CardTech	// Categorias: 00 - DESCONHECIDA() 01 – BARRAS 02 – PROXIMIDADE 03 – SMARTCARD
B [1]	CardType	// Tipos: 01 – EMPREGADO 02 – TERCEIRO 03 – PARCEIRO 04 - VISITANTE, GRUPO DE VISITANTES 05 – OUTRA UNIDADE 06 – PROVISÓRIO 07 – RESPONSÁVEL PELO ALUNO
B[5]	CardID	// Id do crachá no formato hexadecimal, exemplo id decimal = 123456789, id hexadecimal = "00075BCD15" Acesso contém 5 bytes e Relógio 8 bytes.
B[]	Template	//Template, formato binário, pois é uma imagem
B[1]	TamplateFactory	//Fabricante do template 1–Sagem 2–Geomok 3–IR 4–LG 5–Suprema
D	Date	// Data dd/mm/aaaa hh:mm:ss
I[4]	InitTime	// Tempo inicial em minutos a partir da 00 hora Compreendido ente 0000(00:00) e 1439(23:59).
I[4]	EndTime	// Tempo final em minutos a partir da 00 hora

I[4]	InitReservation	// Tempo inicial da reserva em minutos a partir da 00 hora
I[4]	EndReservation	// Tempo final da reserva em minutos a partir da 00 hora
I[4]	InitConsume	// Tempo inicial no refeitório em minutos a partir da 00 hora
I[4]	EndConsume	// Tempo final no refeitório em minutos a partir da 00 hora
I[4]	PermissionCode	// Código da permissão
S[43]	ActivatedReaders	// Leitoras ativas "1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1"
I [2]	BioConfLevel	// Nível de conferencia biométrico da leitora, retornar de 1-100
B[1]	SnackTimeType	// Tipos de dia: 0 – Normal 1– Sábado 2 – Feriado 3–Domingo
S[6]	Pwd	//Senha, apenas números
I[1]	OperationType	//Tipo da operação 0 - Primeira carga ou sobrepor existente 1 - Inclusão de novo registro 2 - Exclusão de registro 3 – Atualiza Cartão ou Biometria (Relógio)
S [x]	PersonID	// Identificador da pessoa Obs:Caso desejar coletar biometria utilizando o teclado ou acesso via teclado: Deve conter apenas números; O identificador terá apenas 16 caracteres, preenchendo a esquerda com 7 espaços(" ") .Ex : " 0000000123456789" . Acesso deve conter 23 bytes e Relógio 30 bytes.
L[]	Pis	//Número do Pis
S[]	Name	//Nome da pessoa, mínimo de 1 e máximo de 52.

I[1]	TypeOfValidation	//Tipo de validação biométrica
		1 = 1:1, utiliza o id físico do cartão para localizar a pessoa e em segundo passo localiza o template armazenado no mapa do cartão Mifare, caso não exista efetua a marcação de ponto, pois a primeira validação com cartão foi validada.
		2 = 1:N, pesquisa a biometria no banco de dados do módulo Biométrico, se localizado procura na lista de empregados para pegar PIS e Nome.
		3 = 1:1, utiliza o id físico do cartão para localizar a pessoa e apartir do resultado, se existir biometria solicita ao módulo biométrico efetuar a pesquisa no no banco de dados do módulo, caso não localize efetua a marcação de ponto, pois a primeira validação com cartão foi validada.

Retorno do envio de Comandos:

0 = Comando Enviado com Successo ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:

Não há dados de retorno

Exemplo C#:
Envio de Comando:

```
//Lista de Permissões
PermissionListReaderPermissionTime[] perLstRdPermTime = new
PermissionListReaderPermissionTime[2];
PermissionListReaderPermission[] perLstRdPermss = new
PermissionListReaderPermission[2];
PermissionListReader[] perLstReader = new PermissionListReader[1];
PermissionList permLst = new PermissionList();

          perLstRdPermTime[0] = new PermissionListReaderPermissionTime();
          perLstRdPermTime[0].DeviceID = 1;
          perLstRdPermTime[0].InitTime = 0001;
          perLstRdPermTime[0].EndTime = 0020;
          perLstRdPermTime[1] = new PermissionListReaderPermissionTime();
          perLstRdPermTime[1].DeviceID = 1;
          perLstRdPermTime[1].InitTime = 0003;
          perLstRdPermTime[1].EndTime = 0040;

          perLstRdPermss[0] = new PermissionListReaderPermission();
          perLstRdPermss[0].PermissionCode = 1;
          perLstRdPermss[0].DeviceID = 1;
          perLstRdPermss[0].Time = perLstRdPermTime;
          perLstRdPermss[1] = new PermissionListReaderPermission();
          perLstRdPermss[1].PermissionCode = 0002;
          perLstRdPermss[1].DeviceID = 1;
          perLstRdPermss[1].Time = perLstRdPermTime;

          perLstReader[0] = new PermissionListReader();
          perLstReader[0].ReaderCode = 1;
          perLstReader[0].DeviceID = 1;

          perLstReader[0].Permission = perLstRdPermss;
          permLst.Readers = perLstReader;

          permLst.DeviceID = 1;

LoadList loadl = new LoadList();
          loadl.DeviceID = 1;
          loadl.CommStatus = 2;
          loadl.SeqCmd = 100;
          loadl.Type = 1;
          loadl.OperationType = 0 ;
          loadl.List = permLst;

          if (DFS.Execute(loadl) < 0) { }
else
          {
              Console.WriteLine(" Comando enviado com sucesso");
}

//Lista de Feriados
HolidaysList holLst = new HolidaysList();
HolidaysListValues[] holValue = new HolidaysListValues[4];  //tamanho da lista
          holValue[0] = new HolidaysListValues();
          holValue[0].Date = "01/01/2012 00:00:00";
          holValue[1] = new HolidaysListValues();
          holValue[1].Date = "02/02/2012 00:00:00";
```

```
         holValue[2] = new HolidaysListValues();
         holValue[2].Date = "03/01/2012 00:00:00";
         holValue[3] = new HolidaysListValues();
         holValue[3].Date = "04/02/2012 00:00:00";
         holLst.Values = holValue;

LoadList loadl = new LoadList();
         loadl.DeviceID = 1;
         loadl.CommStatus = 2;
         loadl.SeqCmd = 10;
         loadl.Type = 2;
         loadl.OperationType = 0 ;
         loadl.List = holLst;

         if (DFS.Execute(loadl) < 0) { }
         else
              {
Console.WriteLine("Comando enviado com sucesso");
}

//Lista de Acessos
AccessList accessLst = new AccessList();
AccessListValues[] accessValue = new AccessListValues[2];
         accessLst.DeviceID = 1;
          // Primeira pessoa
         accessValue[0] = new AccessListValues();
         accessValue[0].ActivatedReaders =
"1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1";
         accessValue[0].CardType = 1;
         accessValue[0].CardID = new Byte[] { 0x01, 0x02, 0x03, 0x04, 0x05 };
         accessValue[0].DeviceID = 1;
         // Segunda pessoa
         accessValue[1] = new AccessListValues();
         accessValue[1].ActivatedReaders =
"1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1";
         accessValue[1].CardType = 1;
         accessValue[1].CardID = new Byte[] { 0x01, 0x02, 0x03, 0x04, 0x05 };
         accessValue[1].DeviceID = 1;

         accessLst.Values = accessValue;

LoadList loadl = new LoadList();
         loadl.DeviceID = 1;
         loadl.CommStatus = 2;
         loadl.SeqCmd = 10;
         loadl.Type = 3;
         loadl.OperationType = 0 ;
         loadl.List = accessLst;

if (DFS.Execute(loadl) < 0) { }
else
            {
Console.WriteLine(" accessLst Comando enviado com sucesso");
            }

//Lista de templates
TemplateListPersonTemplate[] tmpLstPerTemp = new TemplateListPersonTemplate[2];
TemplateListPersonCard[] tmpLstPersCard = new TemplateListPersonCard[1];
TemplateListPerson[] tmpLstPers = new TemplateListPerson[1];
         TemplateList tmplLst = new TemplateList();

         tmplLst.DeviceID = 1;
```

```
//biometria 1
       tmpLstPerTemp[0] = new TemplateListPersonTemplate();
       tmpLstPerTemp[0].DeviceID = 1;
       tmpLstPerTemp[0].TamplateFactory = 1;
       tmpLstPerTemp[0].Template = new Byte[1024];

       //biometria 2
      tmpLstPerTemp[1] = new TemplateListPersonTemplate();
      tmpLstPerTemp[1].DeviceID = 1;
      tmpLstPerTemp[1].TamplateFactory = 1;
      tmpLstPerTemp[1].Template = new Byte[1024];


         tmpLstPersCard[0] = new TemplateListPersonCard();
         tmpLstPersCard[0].CardID = new Byte[] { 0x01, 0x02, 0x03, 0x04, 0x05 };
         tmpLstPersCard[0].DeviceID = 1;
         tmpLstPersCard[0].CardTec = 1;

         tmpLstPers[0] = new TemplateListPerson();
         tmpLstPers[0].BioConfLevel = 1;
         tmpLstPers[0].DeviceID = 1;
         tmpLstPers[0].Card = tmpLstPersCard;
         tmpLstPers[0].Template = tmpLstPerTemp;
         tmpLstPers[0].PersonID = new string('9999', 23);

         tmplLst.Person = tmpLstPers;

       LoadList loadl = new LoadList();
         loadl.DeviceID = 1;
         loadl.CommStatus = 2;
         loadl.SeqCmd = 100;
         loadl.Type = 4;
         loadl.OperationType = 0 ;
         loadl.List = tmplLst;

         if (DFS.Execute(loadl) < 0) { }
            else
            {
Console.WriteLine(" Comando enviado com sucesso");
}

//Lista de Bloqueio
BlockList blockLst = new BlockList();
AccessListValues[] accessValue = new AccessListValues[2];
         blockLst.DeviceID = 1;
         // Primeira pessoa
         accessValue[0] = new AccessListValues();
         accessValue[0].ActivatedReaders =
"1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1";
         accessValue[0].CardType = 1;
         accessValue[0].CardID = new Byte[] { 0x01, 0x02, 0x03, 0x04, 0x05 };
         accessValue[0].DeviceID = 1;
         // Segunda pessoa
         accessValue[1] = new AccessListValues();
         accessValue[1].ActivatedReaders =
"1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1";
         accessValue[1].CardType = 1;
         accessValue[1].CardID = new Byte[] { 0x01, 0x02, 0x03, 0x04, 0x05 };
         accessValue[1].DeviceID = 1;

         blockLst.Values = accessValue;
```

```
LoadList loadl = new LoadList();
         loadl.DeviceID = 1;
         loadl.CommStatus = 2;
         loadl.SeqCmd = 10;
         loadl.Type = 5;
        loadl.OperationType = 0 ;
         loadl.List = blockLst;
        if (DFS.Execute(loadl) < 0) { }
else
           {
               Console.WriteLine(" accessLst Comando enviado com sucesso");
}

//Lista de Empregados
EmployeeList empLst = new EmployeeList();
EmployeeListPerson[] empLstPers = new EmployeeListPerson[2];
EmployeeListPersonCard[] empLstPCard = new EmployeeListPersonCard[2];
EmployeeListPersonCard[] empLstPCard1 = new EmployeeListPersonCard[2];

       empLst.DeviceID = 1;
          // Primeiro empregado
         empLstPers[0] = new EmployeeListPerson();
       empLstPers[0].BioConfLevel = 0;
         empLstPers[0].PersonID = new string('9999', 23);

         empLstPCard[0] = new EmployeeListPersonCard();
         empLstPCard[1] = new EmployeeListPersonCard();

         empLstPers[0].Card = empLstPCard;
         empLstPers[0].Card[0].CardID = new Byte[] { 0x01,0x02,0x03,0x04,0x05 };
         empLstPers[0].Card[0].CardTec = 2;
         empLstPers[0].Card[1].CardTec = 2;
         empLstPers[0].Card[1].CardID = new Byte[] { 0x01,0x02,0x03,0x04,0x06 };

// Segundo empregado
         empLstPers[1] = new EmployeeListPerson();
empLstPers[1].BioConfLevel = 0;
         empLstPers[1].PersonID = new string('x', 23);

         empLstPCard1[0] = new EmployeeListPersonCard();
         empLstPCard1[1] = new EmployeeListPersonCard();

         empLstPers[1].Card = empLstPCard1;
         empLstPers[1].Card[0].CardID = new Byte[] { 0x01,0x02,0x03,0x04,0x07 };
         empLstPers[1].Card[0].CardTec = 2;
         empLstPers[1].Card[1].CardTec = 2;
         empLstPers[1].Card[1].CardID = new Byte[] { 0x01,0x02,0x03,0x04,0x08 };

         empLst.Person = empLstPers;

LoadList loadl = new LoadList();
         loadl.DeviceID = 1;
         loadl.CommStatus = 2;
         loadl.SeqCmd = 10;
         loadl.Type = 6;
         loadl.OperationType = 0 ;
         loadl.List = empLst;
         if (DFS.Execute(loadl) < 0) { }
            else
            {
Console.WriteLine("SnackTimeList ,Comando enviado com sucesso");
```

```
            }
//Lista de Refeitorio
SnackTimeList snkLst = new SnackTimeList();
SnackTimeListValues[] snkValue = new SnackTimeListValues[4];
SnackTimeListReservation[] scnkReser = new SnackTimeListReservation[2];
SnackTimeListReservation[] scnkReserOthers = new SnackTimeListReservation[1];

         snkLst.DeviceID = 1;

          scnkReser[0] = new SnackTimeListReservation();
          scnkReser[0].InitConsume =0004;
          scnkReser[0].EndConsume = 0005;
          scnkReser[0].InitReservation = 0006;
          scnkReser[0].EndReservation = 0007;
          scnkReser[0].DeviceID = 1;

          scnkReser[1] = new SnackTimeListReservation();
          scnkReser[1].InitConsume = 0008;
          scnkReser[1].EndConsume = 0039;
          scnkReser[1].InitReservation = 0010;
          scnkReser[1].EndReservation = 0071;
          scnkReser[1].DeviceID = 1;

          scnkReserOthers[0] = new SnackTimeListReservation();
          scnkReserOthers[0].InitConsume =0000;
          scnkReserOthers[0].EndConsume = 0001;
          scnkReserOthers[0].InitReservation = 0000;
          scnkReserOthers[0].EndReservation = 0001;
          scnkReserOthers[0].DeviceID = 1;

          snkValue[0] = new SnackTimeListValues();
          snkValue[0].DayType = 0; // Normal
          snkValue[0] = new SnackTimeListValues() ;
          snkValue[0].Reservation = scnkReser;

          snkValue[1] = new SnackTimeListValues();
          snkValue[1].DayType = 1; // Sábado
          snkValue[1] = new SnackTimeListValues() ;
          snkValue[1].Reservation = scnkReserOthers;

          snkValue[2] = new SnackTimeListValues();
          snkValue[2].DayType = 2; // Feriado
          snkValue[2] = new SnackTimeListValues() ;
          snkValue[2].Reservation = scnkReserOthers;

          snkValue[3] = new SnackTimeListValues();
          snkValue[3].DayType = 3; // Domingo
          snkValue[3] = new SnackTimeListValues() ;
          snkValue[3].Reservation = scnkReseOthersr;


          snkLst.Values = snkValue;

LoadList loadl = new LoadList();
          loadl.DeviceID = 1;
          loadl.CommStatus = 2;
          loadl.SeqCmd = 10;
          loadl.Type = 7;
          loadl.OperationType = 0 ;
          loadl.List = snkLst;
```

```
         if (DFS.Execute(loadl) < 0) { }
else
            {
               Console.WriteLine("SnackTimeList ,Comando enviado com sucesso");
}


//Lista de senhas
PwdList pwdLst = new PwdList();
PwdListValues[] pwValue = new PwdListValues[3];
LoadList loadLst = new LoadList();

       pwValue[0] = new PwdListValues();
       pwValue[0].CardID = new Byte[] { 0x01, 0x02, 0x03, 0x04, 0x05 };
       pwValue[0].DeviceID = 1;
       pwValue[0].Pwd = "123456";

       pwValue[1] = new PwdListValues();
       pwValue[1].CardID = new Byte[] { 0x00, 0x02, 0x03, 0x04, 0x00 };
       pwValue[1].DeviceID = 1;
       pwValue[1].Pwd = "432156";

       pwValue[2] = new PwdListValues();
       pwValue[2].CardID = new Byte[] { 0x00, 0x02, 0x03, 0x04, 0x00 };
       pwValue[2].DeviceID = 1;
       pwValue[2].Pwd = "227788";

       pwdLst.DeviceID = 1;
       pwdLst.Values = pwValue;

       loadLst.CommStatus = 2;
       loadLst.DeviceID = 1;
       loadLst.SeqCmd = 123;
       loadLst.List = pwdLst;
       loadl.OperationType = 0 ;

       loadLst.Type = 8;

         if (DFS.Execute(loadLst) < 0) { }
else
            {
                Console.WriteLine(" accessLst Comando enviado com sucesso");
            }

//Lista de Empregados Relógio, esta lista é enviada de um-a-um pela APP é
unitária para o DFS

EmployeeListClock employeeListsClock = new EmployeeListClock();
       Person person = new Person();
       CardClock[] cardClock = new CardClock[1];
       Templates[] templates = new Templates[2];

       cardClock[0] = new CardClock();
       cardClock[0].CardTec = 3;
       cardClock[0].CardID = new byte[]{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x01};
       person.CardClock = cardClock;

       templates[0] = new Templates();
       templates[0].Template = new byte[300];
       templates[1] = new Templates();
       templates[1].Template = new byte[300];
       person.Templates = templates;
```

```
        person.Name = "Jesus";
        person.PersonID = "123456789".PadRight(30,' ');
        person.Pis = 123456789;
        person.RecordKeyboard = 1;
        person.TypeOfValidation = 2;

        employeeListsClock.Person = person;

        LoadList loadList = new LoadList();
        LoadList.DeviceID = Convert.ToInt32(lstB1.SelectedItem.ToString());
        loadList.CommStatus = 2;
        loadList.SeqCmd = 10;
        loadList.Type = 9;
        loadList.List = employeeListsClock;
        loadList.OperationType = 0;

        if (DFS.Execute(loadList) < 0) { }
             else
             {
                 Console.WriteLine("EmployeeListClock, Comando enviado com
        sucesso");
             }
```

Recepção de Resposta Assincrona Acesso:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
   {
   String BuffMsg = "";
   if (args.Msg is LoadListResp)
   {
      DFS.Ack(args.Msg.DeviceID);
BuffMsg =
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID " + ((LoadListResp)args.Msg).DeviceId.ToString() +
"\nSequencia CMD " + ((LoadListResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((LoadListResp)args.Msg).ReturnCode.ToString() +
"\nData Ger CMD  " + ((LoadListResp)args.Msg).CmdGenerationTimestamp.ToString() +
"\nData Exe CMD  " + ((LoadListResp)args.Msg).CmdExecutionTimestamp.ToString()  +
"\nTam dos Dados " + ((LoadListResp)args.Msg).PayloadSize.ToString();

Console.WriteLine(BuffMsg );
 }
}
```

Recepção de Resposta Assincrona Relógio

```
String BuffMsg="";
   //OperationType = 0
if (args.Msg is InsertUpdatePersonResp)
            {
               BuffMsg = "Retorno do comando InsertUpdatePerson " +
               "\nDeviceID " +
((InsertUpdatePersonResp)args.Msg).DeviceID.ToString() +
                "\nPersonID " + ((InsertUpdatePersonResp)args.Msg).PersonID +
                "\nErrorCount " +
((InsertUpdatePersonResp)args.Msg).ErrorCount.ToString() +
                "\nErrors : \n";

                for (int i = 0; i <
((InsertUpdatePersonResp)args.Msg).ErrorCount; i++)
                {
```

```
BuffMsg += (ClockResult)((InsertUpdatePersonResp)args.Msg).ErrorCodeList[i] +
"\n";
                }

                DFS.Ack(args.Msg.DeviceID);

                                Console.WriteLine(msg );

            }
   //OperationType = 3
            else if (args.Msg is InsertUpdateReadersResp)
            {
                BuffMsg = "\nRetorno do comando InsertUpdateReaders " +
               "\nDeviceID " +
((InsertUpdateReadersResp)args.Msg).DeviceID.ToString() +
               "\nPersonID " + ((InsertUpdateReadersResp)args.Msg).PersonID +
               "\nErrorCount " +
((InsertUpdateReadersResp)args.Msg).ErrorCount.ToString() +
                "\nErrors : \n";

                for (int i = 0; i <
((InsertUpdateReadersResp)args.Msg).ErrorCount; i++)
                {
                    BuffMsg +=
(ClockResult)((InsertUpdateReadersResp)args.Msg).ErrorCodeList[i] + "\n";
                }

                DFS.Ack(args.Msg.DeviceID);

                consoleLog.BeginInvoke(new updateResp(AtualizaResp), new object[]
{ msg });

            }
   //OperationType = 2
            else if (args.Msg is DeletePersonResp)
            {
                BuffMsg = "Retorno do comando DeletePersonResp " +
               "\nDeviceID " + ((DeletePersonResp)args.Msg).DeviceID.ToString() +
               "\nPersonID " + ((DeletePersonResp)args.Msg).PersonID +
               "\nStatus " + (ClockResult)((DeletePersonResp)args.Msg).Status
               ;

                DFS.Ack(args.Msg.DeviceID);

                Console.WriteLine(BuffMsg );

            }
```

9.5.6. Exclusão de Listas – DeleteList()

Descrição:
Esta função deleta uma lista do dispositivo indicado previamente carregada.
A lista 4 (Templates) não é excluída porque possuí relação com a lista de empregados, sua exclusão será realizada quando a lista 6 (Empregados) for excluída.

Função:
Int Execute(DeleteList)

Parâmetros:

I [1]	CommStatus	// Estado da comunicação, 1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando

B [1]	Type	// Lista de Permissões = 1 ,
		// Lista de Feriados = 2,
		// Lista de Acesso = 3,
		// Lista de Bloqueios = 5
		// Lista de Refeitorio = 7
		// Lista de Senhas = 8
		// Lista de empregados para Ponto = 9 (**implementação futura**)

Dados da Resposta do comando:
Não há dados de retorno

Exemplo C#:
Envio de Comando:

```
DeletList obj = new DeletList();
              obj.Type = 2;
              obj.CommStatus = 2;
              obj.DeviceID = 1;
              obj.SeqCmd = 2;
         if (DFS.Execute(loadLst) < 0) {
              }
else
            {
Console.WriteLine(" accessLst Comando enviado com sucesso");
            }
```

Recepção de Resposta Assincrona:

```
 if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
{
  String BuffMsg = "";
   if (args.Msg is DeletListResp)
    {
     DFS.Ack(args.Msg.DeviceID);
BuffMsg =
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID      " + ((DeletListResp)args.Msg).DeviceId.ToString() +
"\nSequencia CMD " + ((DeletListResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((DeletListResp)args.Msg).ReturnCode.ToString() +
"\nData Ger CMD " + ((DeletListResp)args.Msg).CmdGenerationTimestamp.ToString() +
"\nData Exe CMD " + ((DeletListResp)args.Msg).CmdExecutionTimestamp.ToString()  +
"\nTam dos Dados " + ((DeletListResp)args.Msg).PayloadSize.ToString();

Console.WriteLine(BuffMsg );
  }
}
```

9.6. Comandos de Dispositivos de Acesso

Entende-se sob comando as ações executadas pela APP que através do DFS são validadas e enviadas ao FWR, cujas respostas podem ser Síncronas e Assíncronas.
Nesta sessão serão listados os comandos comuns aos dipositivos de Ponto e Acesso.

O retorno dos comandos é realizado de forma Síncrona, caso a execução seja “imediata”. Se a execução for demorada, será enviado o retorno em forma de evento, porém o retorno imediato ainda acontecerá indicando que o firmware conseguiu entender o comando ou não, ou seja, todo comando possui uma resposta Síncrona no mínimo.

9.6.1. Carga da Configuração do Firmware – ConfigFirmware()

Descrição:
Esta função executa comando de configuração de um determinado dispositivo.
Muita atenção na codificação desta função, pois ela contém uma quantidade de bytes que aproxima 20 K (20480), mesmo que uma estrutura não for utilizada deverá ser preenchida com zeros ou valores pertinentes aos tipos dos dados.

Função:
Int Execute(ConfigFirmware);

Parâmetros:

I[1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I[9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I[9]	SeqCmd	// Sequencial do comando
I[9]	newDeviceID	//ID do dispositivo desejado, após configuração=DeviceID
I[1]	ConnMode	//Tipo de endereço IP, 0=IP fixo, 1=DHCP
N[15]	newDeviceIP	//Endereço IP desejado para o Device ex: “172.16.0.23”
N[15]	newNetMask	//Mascara de sub-redeex: ”255.255.255.0”
N[15]	newNetGw	//Gateway do device desejado ex:”172.16.0.23”
N[15]	serverIP	//Endereço IP do servidor ex:”172.16.0.23”
I[5]	serverPort	//Endereço da Porta do servidor ex:”3232”
D	InitSummerTime	//Data inicial do horário de verão ex:”dd/mm/aaaa ”
D	EndSummerTime	//Data final do horário de verão ex:”dd/mm/aaaa ”
I[3]	UTC	//Tempo universal coordenado em minutos, ex: 180 (3 horas), -180 (-3)
I[2]	SubsidiaryID	//ID da filial ex:”identificador da empresa”
S[16]	StandardMsg	//Mensagem padrão do dispostivo para aplicação, exemplo nome do departamento ou área que se encontra o controlador, “Almoxarifado-1”, “Datacenter”, “Sala-1”, “2 Andar”, “Refeitorio”, etc.
I[1]	UseDisplay	//Identificador do display do dispositivo, 1 = usa, 0 = não usa. Caso for uma MCANet como master de MRA, as leitoras devem estar como 0.
I[1]	KeyBlock	// Habilita poder efeturar o bloqueio teclado do dispositivo manualmente, 1=permite, 0=não permite. MCA (enter + anula / anula + enter), MCANet (enter + esc / esc + enter).
I[4]	CardExpirationDays	//Validador de dados dos cartões em dias
I[5]	KeepAlive	//Tempo do KeepAlive em milissegundos, mínimo 3 segundos e máximo 60 segundos o default é 5 segundos. É utilizado este tempo para cálculo randômico de envio de pedido de nova conexão.
I[5]	CommTimeout	//Timeout de comunicação DFS<>FWR emdecissegundos, este tempo observado nos limites dos requisitos funcionais, deve

		ser calibrado de tal foram que consiga responder a contento as requisições enviadas pelo FWR.
T	**Inicio Funções x 10**	//Funções programáveis para teclado. Atenção! A tecla “0” corresponde a função “1” é de uso do sistema e é reservada.
I[1]	Type	//Tipo da função solicitada pelo device, 1 - Cadastro de templates biométricos; 2 - Ignora controle de Nível; 3 - Digitação de cadastro para acesso; 4 - Reserva de refeição; 5 - Cancela reserva de refeição; 6 - Ativa Modo de Emergência.
S[16]	ActivationMsg	//Mensagem de ativação da função solicitada, mostrada quando a tecla é pressionada.
S[16]	ConfirmationMsg	//Mensagem de confirmação da mensagem.

A[3]	ExeActuator	//Executa acionamentos e Cadastro de biometria com autorização do cartão mestre, estrutura texto (b1;b2), formato "0;0".
		b1=executa Acionamentos : 1=sim, 0=não, **ATENÇÃO**, caso seja configurado qualquer acionamento nesta controladora a primeira função = 1 deverá esta aqui assinalado com valor 1.
		b2=cadastro de biometria exige cartão mestre? : 1=sim, 0=não.

S[16]	PersonalRecordMask	//Máscara para registro de pessoa via teclado, quando a pessoa realiza sua marcação via teclado, é possível configurar uma máscara para a digitação. Esta máscara deve possuir 16 caracteres compostos por: I (Tipo da pessoa), E (Empresa), T (Tipo do colaborador (Se existir tipo de colaborador na máscara o tipo de pessoa será descartado)), C (Cadastro da pessoa) Exemplo: EEEETTCCCCCCCCCC, IIEECCCCCCCCCCCC, Para pré-inicializar a máscara. Ex.: 01EE020012CCCC00. Esta máscara é utilizada para seqüenciar o get no display das informações desta máscara, o DFS validará o conteúdo deste campo para apenas permitir “IETC” e “dígitos” que será considerado como parte fixa na montagem do código final

I[1]	IgnoreLevel	//Ignora o nível de acesso,
		1=sim, 0= não, ao contrário em uma reserva de refeitório será negado nível.

T	**Fim Funções x 10**	
T	**Inicio Inputs x 4**	
I [1]	StandardOn	//Estado padrão, 1=ligado, 0=desligado, este parâmetro é análogo a “NF/NA”, ou seja, circuito normalmente aberto ou fechado. No caso de um MRA: a entrada 1 corresponde a botão e a entrada 2 corresponde a sensor. MCANet e MRA, utilizam apenas as duas primeiras entradas.

I [1]	Monitored	//Monitora, 1=sim, 0=não, monitora ou não para alarmes, podendo ser utilizada como entrada dependente para um acionamento
I [1]	Delay	//Delay de ativação em milissegundos, quanto tempo depois para ser gerado um evento
T	**Inicio Dias Úteis x 7**	//Exclusões para o acionamento durante os dias úteis
S [4]	BusinessMaskInit	//Faixa inicialde minutos decorridos da zero hora até 23:59
S [4]	BusinessMaskEnd	//Faixa final de minutos decorridos da zero hora(0000 minuto) até 23:59(1349 minutos)
T	**Fim Dia Úteis x 7**	
T	**Inicio Sábados x 7**	//Exclusões para o acionamento durante os sábados
4]	SaturdayMaskInit	//Faixa inicial de minutos decorridos da zero hora(0000minuto) até 23:59(1349 minutos).
S [4]	SaturdayMaskEnd	//Faixa final de minutos decorridos da zero hora(0000 minuto) até 23:59(1349 minutos).
T	**Fim Sábados x 7**	
T	**Inicio Domingos x 7**	//Exclusões para o acionamento durante os domingos
S [4]	SundayMaskInit	//Faixa inicial de minutos decorridos da zero hora(0000minuto) até 23:59(1349 minutos)
S [4]	SundayMaskEnd	//Faixa final de minutos decorridos da zero hora(0000 minuto) até 23:59(1349 minutos).
T	**Fim Domingos x 7**	
T	**Inicio Feriados x 7**	//Exclusões para o acionamento durante os feriados
S [4]	HolidayMaskInit	//Faixa inicial de minutos decorridos da zero hora(0000minuto) até 23:59(1349 minutos).
S [4]	HolidayMaskEnd	//Faixa final de minutos decorridos da zero hora(0000 minuto) até 23:59(1349 minutos).
T	**Fim Feriado x 7**	
T	**Fim Inputs x 4**	
A [25]	StandardDigitalOut	//Estado padrão das saídas digitais, 1 byte para cada saída, 1=ativado, 0=desativado, formato “0;0;0;0;0;0;0;0;0;0;0;0;0”
I [4]	BurlaTime	//Tempo de burla da catraca em milissegundos

I [1]	RfidType	//Tipo de leitora de RFID:
		1=HID34
		2=HID26
		3=ACURA26
		4=WIEGAND26
		5=CLOCKDATA
		6=ACURA_ABATRACK
		7=INDALA_ABATRACK
		8=ICLASS_MIFARE

		9=ICLASS_WIEGAND37 10=ICLASS_TEXAS64 11=WIEGAND30 Quando for configurada uma leitora RFID, é obrigatório definir o tipo da leitora.

I[1]	InitParityBit	//Bit de paridade inicial,
		0=não verifica paridade,
		1=verifica paridade,
		2=paridade zero,
		3=paridade um

I [1]	EndParityBit	//Bit de paridade final, 0=não verifica paridade, 1=verifica paridade,2=paridade zero,3=paridade um

I [1]	RfidNumComp	//Composição do número de RFID,
		1=leitura tradicional/somente número físico,
		2=facility code + número físico.

S[24]	BarcodeCripKey	//chave de criptografia para barras, esta chave é fornecida pelo fabricante Telemática quando utilizado o tipo de criptografia (BarcodeCripType)=2
I [1]	BarcodeCripSubtype	//Sub-tipo de criptografia para barras, quando o tipo de criptografia for utilizado (BarcodeCripType)= 2, deverá ser informado qual algoritimo utilizar

I [1]	BarcodeCripType	//Tipo de criptografia para barras:
		0=nenhum
		1=Senior
		2=Telemática
		3=Contax

S [25]	BarcodeMask	//Máscara do código de barras, Ex.: 0000000000000000000CCCCC: será utilizado somente os últimos 5 números do código de barras
S [12]	BadgeMask	// Máscara de composição do crachá (Somente para o controle do SAD da Petrobrás):E(Empresa), F(Filial), I(Identificador de uso do crachá), V(Via do crachá), R ou C(Registro/Número do crachá), Ex.: EEFFICCCCCCCC
I [4]	AccessSendedFileLength	//tamanho do aquivo dos eventos de acesso enviados, defindo em KB, o valor mínimo é 10 KB. A somatória máxima de consumo dos 5 (cinco) arquivos a seguir em bytes é 1572864 Bytes (1.5 MB), portanto a APP deve fazer um balanceamento de uso. O DFS pode barrar descer uma configuração que ultrapassar esse limite. Este arquivo o padrão é 256K.
I [4]	AlarmSendedFileLength	//tamanho do aquivo dos eventos de alarmes enviados, defindo em KB, o valor mínimo é 10 KB padrão 256K.
I [4]	AccessFileLength	//tamanho dos aquivos armazenados no firmware definidos, defindo em KB, o valor mínimo é 10 KB e padrão 256K.
I [4]	AlarmFileLength	//tamanho do aquivo de alarmes, defindo em KB, o valor mínimo é 10 KB e padrão é 256K.
I [4]	LogFileLength	//tamanho doaquivo de log, defindo em KB, o valor mínimo é 10 KB e padrão é 64K.

T **Inicio LEITORASx 22**

Observações:
O firmware permite que sejam configuradas até 22 leitoras, sendo estas para o controle de acesso a mais importante das configurações, onde a partir de uma leitura válida ou não o firmware tomará as ações configuradas nesta sessão, ex: cartão válido abre porta, aciona led verde com beep curto; cartão inválido não abre porta, aciona led vermelho com beep longo.

A configuração é posicional, ou seja, as três primeiras abaixo são SmartCard e deverá obedecer a seguinte ordem:

1,2 e 3 = SmartCard MCA (1=passiva CN24 (cabo coaxial), 2=ativa CN20 e 3=ativa CN21), MCANet (CN4);
4,5 e 6 = RFID MCA (CN1 = Multiplexada), Pinos =(2 e 3), (4 e 5) e (6 e 7), MCANet CN7;
7,8 e 9 = Barras MCA (CN15, CN17, para utilizar a terceira Barras CN12 (eliminando a porta USB)), MCANet CN9;
10,11 e 12 = Biométricas (FingerPrint), usar apenas 10 (Sagem) MCA (CN15 ou CN17), MCANet (CN6);
13 = Teclado MCA (CN27);
14,15 e 16 = Biométricas (HandKey) MCA (CN15 ou CN17);
17 até 22 = MRA (Módulo de Acesso Remoto) ou leitoras remotas, ligadas através de RS 485 utilizando como master apenas a controladora MCANet.

Atenção Para Conflitos:
Verificar utilização das CNs, pois no caso todas precisa ser ligadas em CN e o mesmo CN não pode ser utilizado para mais que uma leitora.

Parâmetros:

I[1]	Active	//Ativar Leitora, 1=sim, 0=não
I[4]	ReaderAdress	//Endereço da leitora. Usado quando for MRA ou futuro para outros tipos dispositivos que necessitem.
I[3]	Level1	//Codificação da leitora que representa o nível 1, atual
I[3]	Level2	//Codificação da leitora que representa o nível 2, seguinte
A [7]	AccessWay	//Direção do acesso das leitoras de acordo com o nível, formato (0;0;0;0), o valor é de acordo com a combinação dos bytes na sua posição.

Esta combinação sempre analisará os valores dos atributos Level1 e Level2, este controle não significa que a controladora terá o controle de Nível propriamente dito, depende do byte indicativo “CheckLevel”.

Sobre Nível 1 e Nível 2, diz respeito ao controle de nível (perímetro), o qual o sistema mantém um valor representando o nível de localização atual da pessoa, dentro da faixa 0 até 98, o valor 99 é reset de resposta permitindo ambos sentidos. Considerando que fora da empresa o nível de localização é 0, e dentro do pátio da empresa o nível de localização é 1, então se configura Level1 com valor 0, e Level2 com valor 1, isto informa que o controle da leitora pode ser no sentido 0 -> 1 ou 1 -> 0, ou seja, se o nível do usuário estiver como 0, passando o cartão nesta leitora, ele estaria tentando acessar o Nível 2 (configurado com valor 1). O uso dos atributos Nível 1 e Nivel 2 pode também estar associado ao controle do sentido do giro, quando utilizamos uma catraca com apenas uma leitora (MCA Plus).

Valor de acordo com a combinação dos bytes a seguir:

byte 1: Leitora Permite Ambos os Sentidos?

[0] Apenas [Nivel1->Nivel2]
[1] [Nivel1->Nivel2] ou [Nivel2->Nivel1]

byte 2: Sentido do giro
[0] Horário
[1] Anti-horário

byte 3: Para envio do sentido do giro nos eventos, ou seja, depois do ocorrido avisar qual sentido foi o giro [Nivel1->Nivel2]
[0] Entrada
[1] Saída

byte 4: Inverso do byte 3, se foi configurado para "0", aqui deverá ser "1" [Nivel2->Nivel1].
[0] Entrada
[1] Saída

I [1]	CheckPermission	//Verifica permissão, 1= sim, 0= não, verifica se o grupo de permissão será validado
I [1]	CheckPermissionTime	//Verifica faixa horária de permissão, 1= sim, 0= não
I[1]	CheckPersonalTime	//Verifica faixa horária de pessoa, 0 = não verifica; 1 = ambos sentidos; 2 = entrada; 3 = saída
I [1]	CheckAbsence	//Verifica afastamento, 1=sim, 0=não
I [1]	CheckSituation	//Verifica situação, 1=sim, 0= não
I [1]	CheckExpiration	//Verifica validade, 1=sim, 0=não
I [1]	CredControl	//Controle de crédito, 0=não controla, 1=somente entrada, 2=somente saída, 3= ambos
I [1]	CheckLevel	//Verifica Nível, 1=sim, 0= não
I [1]	CheckAntiDouble	//Verifica passagem de antidupla, 1= sim, 0= não
I [1]	CheckLunchTime	//Verifica intervalo de almoço, 1=sim,0=não
I [1]	CheckAbsenceBlock	//Verifica bloqueio por falta, 1=sim,0=não
I [1]	CheckWorkday	//Verifica bloqueio por inter-jornada, 1=sim, 0= não
I [1]	CheckPwd	//Verifica senha, 1=todos, 2=conforme pessoa
I[1]	CheckBadgeBlock	//Verifica bloqueio por identificador de uso do crachá, 1=sim, 0=não
I[1]	CheckSubsidiaryBlock	//Verifica bloqueio por filial, 1=sim, 0=não
I[1]	CheckForeignSubsidiaryBlock	//Verifica controle por filial estrangeira, 1=sim, 0=não
I [1]	UseDisplay	//Verifica se o device usa display, 1=sim, 0= não. Se for uma MRA este valor deve ficar como 0, desta forma a master não escreverá mensagens do display que poderá confundir os usuários.

I [1]	BioAssociated	//Leitora biométrica associada para validação 1:1:
		0=sem leitora associada,
		10=leitora sagem porta CN17
		11=não utilizado
		12=não utilizado
		14=leitora HandKey porta CN17
		15=leitoraHandKey porta CN15
		16=não utilizado

I [4]	MaxBioUser	//Quantidade máxima de usuários no leitor biométrico:
		1=500

		2=3000 4=5000

I [2]	BioTec	//Tecnologia biométrica da leitora:
		0=não configurada,
		1=SAGEM
		2=GEOMOK (não liberado)
		3=IR (HandKey)
		4=LG (não liberada)
		5=SUPREMA (não liberada)
		99=OUTRAS.

I [2]	BioConfLevel	//Nível de conferencia biométrico da leitora, retornar de 1-100 % o FWR converterá de acordo com cada tecnologia, exemplo: Sagem 1-7 (parâmetro), caso seja informado 50% o FWR converterá para 3, se 100% para Sagem será 7.
I [1]	ReaderValidate	//Indica se a leitora faz validação 1 x N, 1=sim, 0=não
A [5]	ReaderType	//Tipo da leitora: Estrutura texto (b1;b2,b3), formato “0;0;0" 0;0;b3 = sem tipo 0;1;b3 = acesso e ponto 1;0;b3 = acesso 1;1;b3 = ponto b1;b2;0 = com MRA modo local b1;b2;1 = com MRA modo remoto
A [31]	PersonTypeDir1	//Tipo de pessoa direção1 (Nivel 1), máscara “0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0”: (b1;b2;b3;b4;b5;b6;b7;b8;b9;b10;b12;b13;b14;b15;b16) 0 = permite / 1 = bloqueia todos bytes 0 = sem tipo, b1 empregado, b2 parceiro, b3 visitante e grupo de visitantes, b4 outra unidade, b5 provisório, b6 responsável de aluno, b7 crachá mestre, b8 pacientes, b9 alunos, b10 acompanhante de paciente, b11 autorização de entrada, b12 candidato, b13 todos os tipos
A [31]	PersonTypeDir2	//Tipo de pessoa direção2 (Nivel 2), máscara “0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0”, (b1;b2;b3;b4;b5;b6;b7;b8;b9;b10;b12;b13;b14;b15;b16) 0 = permite / 1 = bloqueia todos bytes 0 = sem tipo, b1 empregado, b2 parceiro, b3 visitante e grupo de visitantes, b4 outra unidade, b5 provisório, b6 responsável de aluno,

		b7 crachá mestre, b8 pacientes, b9 alunos, b10 acompanhante de paciente, b11 autorização de entrada, b12 candidato, b13 todos os tipos
I [1]	CheckList	// Verifica lista local, 0 = não verifica; 1=apenas verifica lista; 2=verifica e exclui registro. Se estiver on-line executará de forma redundante.
I [1]	ShowAccessDirection	//Mostra a direção do acesso, 1=sim, 0=não
B [16]	StayTime	//Tempo mínimo de permanência por tipo depessoa. Array de 16 posições. O tempo em minutos que será utilizado na validação de pemanência para cada tipo de pessoa. Valor de 0 á 255.
B [16]	FriskPercent	//Percentual da revista aleatória por tipo depessoa. Array de 16 posições. É o percentual que o firmware utilizará para sorteio por tipo de pessoa. Valor de 0 á 100.
B [16]	CheckBDCC	//Verifica validação BDCC por tipo de pessoa. Array de 16 posições, 0 ou 1, indica se o tipo da pessoa realizar validação BDCC ou não.

T	**Inicio FAIXAS ALEATÓRIAS x 7**	// Este bloco contem as faixas de horários em que comprenderá o período para revista aleatória.
I [4]	Begin	//Tempo inicial para revista aleatória em minutos a partir de 0000 (00:00) até 1439 (23:59).
I [4]	End	//Tempo final para revista aleatória em minutos a partir de 0000 (00:00) até 1439 (23:59).
T	**Fim FAIXAS ALEATÓRIAS x 7**	

T	**Inicio VALIDAÇÃO DE ENTRADA x 4**	//Este bloco é utilizado especificamente para monitoria da urna, onde após um cartão ser invalidado devido usuário ter que depositar na urna pode-se trocar o evento que esta ocorrendo na entrada (inválido) para um de saída (válido).

O estado default do sensor da urna fica ligado (borda de descida), quando um cartão passa envia sinal para entrada. Para fins de entendimento: quando o sinal muda de 0 para 1, chamamos de borda de subida, e quando o sinal muda de 1 para 0, chamamos de borda de descida. Imagine a entrada em seu estado default, logo, estamos com sinal 0. Quando a entrada fica ativa (sai do estado default), temos sinal 1. Se o tipo de borda estiver configurado como subida, então a ação acontece no momento que a entrada sai do estado default (caso de configuração de Urna). Se o tipo de borda estiver configurado como descida, então a ação acontece no momento que a entrada volta ao estado default (caso de configuração de Porta Eclusa).

I[3]	EnterEventCode	//Código de evento de entrada o que esta em curso, normalmente o inválido
I[3]	ExitEventCode	//Código de evento de saída o que sendo substituído que é válido após ter detectado a presença

I [1]	BordType	//Tipo de borda, 1=descida, 2=subida, o estado default do sensor da urna fica ligado, quando um cartão passa envia sinal para entrada
I[4]	WaitTime	//Em milissegundos,tempo de espera da passage do cartão pelo sensor, caso não depositado neste tempo a mensagem que será exibida é a anteriormente setada como Acesso Negado Pelo Tipo.
T	**Fim VALIDAÇÃO DE ENTRADA x 4**	
T	**Inicio ACÕES DE ACESSO VÁLIDO x 1**	
I[1]	EnterDependence	// Número da entrada (1,2,3 ou 4),dependente para o acionamento, utilizada ao invés de tempo para acionamento
T	**Inicio PICTOGRAMA x 1**	

A[7]	ExitAction	//Ação de saída, estrutura texto (b1;b2;b3;b4),
		b1 (1= tem acionamento e 0= não tem);
		b2 (1= liga seta anti horária e 0 = desliga);
		b3 (1=liga seta horária e 0= desliga) e
		b4 (1= ligaX e 0 = desligar X)
		// Caso controladora MCANet
		"1;0;0;1" = vermelho
		"1;1;0;0" = verde
		"1;1;0;1" = verde e vermelho

I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim,0=não
T	**Fim PICTOGRAMA x 1**	
T	**Inicio BUZZER x 1**	
I [1]	ExitAction	//Ação de saída, 1=ligar, 0=desligar
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim,0=não
T	**Fim BUZZER x 1**	
T	**Inicio URNA x 1**	// Na validação do acesso se estiver configurado é ativado o acionamento da urna mesmo com erro de leitura do cartão smartcard.
I [1]	ExitAction	//Ação de saída, 1=ligar, 2=desligar
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim URNA x 1**	
T	**Inicio SOLENOIDES x 2**	
I [1]	ExitAction	//Ação de saída, 1=ligar, 0=desligar Se MRA, "Solenóide 1" funciona como configuração de ação da saída para TRANSISTOR.
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos Se MRA, é o tempo de acionamamento em milissegundos para TRANSISTOR (Solenóide 1).
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não Não usado em MRA.
T	**Fim SOLENOIDES x 2**	

T **Inicio SAÍDASx 8**

I [1] ExitAction //Ação de saída, 1=ligar, 2=desligar
Se MRA, "Saída 1" funciona como configuração de ação da saída para RELÉ.

I [4] ActionTime //Tempo do acionamento da saída em milissegundos
Se MRA, é o tempo de acionamamento em milissegundos para RELÉ (Saída 1).

I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não
Não usado em MRA.

T **Fim SAÍDASx 8**

T **Inicio PICTOGRAMA A DIREITA x 1**

A [7] ExitAction //Ação de saída, estrutura texto (b1;b2;b3;b4),
formato "0;0;0;0" .
b1 (1 = tem acionamento e 0 = não tem);
b2 (1 = liga seta anti horária e 0 = desliga);
b3 (1 =liga seta horária e 0 = desliga) e
b4 (1 = liga X e 0 = desligar X)

I [4] ActionTime //Tempo do acionamento da saída em milissegundos

I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não

T **Fim PICTOGRAMA A DIREITA x 1**

T **Inicio PICTOGRAMA A ESQUERDA x 1**

A [7] ExitAction //Ação de saída, estrutura texto (b1;b2;b3;b4),
formato "0;0;0;0" .
b1 (1 = tem acionamento e 0 = não tem);
b2 (1 = liga seta anti horária e 0 = desliga);
b3 (1 =liga seta horária e 0 = desliga) e
b4 (1 = liga X e 0 = desligar X)

I [4] ActionTime //Tempo do acionamento da saída em milissegundos

I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não

T **Fim PICTOGRAMA A ESQUERDA x 1**

T **Inicio CATRACA x 1** // Caso configurar catraca não terá efeito as estrutruras:
Pictograma, Entradas 1 e 2 (monitora giro) e
Solenóides 1 e 2 (trava sentido).

A [7] ExitAction //Ação de saída, estrutura texto (b1;b2;b3;b4),
b1 á b3 valores 0 ou 1 .Ex ("0;0;0;0")
"**1**;0;b3;b4" b1 (1=sentido horário);
"0;**1**;b3;b4" b2 (1=sentido anti-horário);
"**1**;**1**;b3;b4" b1,b2(1=ambos sentidos) ;
"b1;b2;**1**;b4" b3 (1=combinações de giros anteriores porém para CLIP);
"b1;b2;b3;**1**" b4 (1=combinações de giros anteriores porém para MCANet).

I [4] ActionTime //Tempo do acionamento da saída em milissegundos

I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não

T **Fim CATRACA x 1**

T **Inicio DISPLAY x 1**

I [1] ExitAction //Ação de saída, valor 1=mostrar/ o resto não faz nada

I [4] ActionTime //Tempo do acionamento da saída em milissegundos

I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não
T **Fim DISPLAY x 1**

S[16] DisplayPrimeiraLinha //Texto da primeira linha do display, prencher com zeros para não exibir texto
S [16] DisplaySegundaLinha //Texto da segunda linha do display, preenchercom zeros para não exibir texto

T **Inicio SINALIZAÇÃO DE LEITORA x 1** //Utiliza esta configuração para leitoras externas do tipo Ativas que possuem sinalizadores de leds e buzzer
I [1] Action //Ação,
1=apenas buzzer,
2 =apenasledverde,
3=buzzer e led verde,
4=apenas ledvermelho,
5=buzzer e led vermelho
I [4] BuzzerTime //Tempo de acionamento do buzzer em milissegundos
I [4] LedTime //Tempo de acionamento do led em milissegundos
I [1] EnterDependence //Acionamento depende de entrada, 1=sim, 0=não
T **Fim SINALIZAÇÃO DE LEITORA x 1**

T **Fim ACÕES DE ACESSO VÁLIDO x 1**

T **Inicio ACÕES DE ACESSO NEGADO x 1**
I [1] EnterDependence // Número da entrada (1,2,3 ou 4), dependente para o acionamento, utilizada ao invés de tempo para acionamento

T **Inicio PICTOGRAMA x 1**
A[7] ExitAction //Ação de saída, estrutura texto (b1;b2;b3;b4),
b1 (1= tem acionamento e 0= não tem);
b2 (1= liga seta anti horária e 0 = desliga);
b3 (1=liga seta horária e 0= desliga) e
b4 (1= ligaX e 0 = desligar X)
// Caso controladora MCANet
"1;0;0;1" = vermelho
"1;1;0;0" = verde
"1;1;0;1" = verde e vermelho
I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim,0=não
T **Fim PICTOGRAMA x 1**

T **Inicio BUZZER x 1**
I [1] ExitAction //Ação de saída, 1=ligar, 0=desligar
I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim,0=não
T **Fim BUZZER x 1**

T **Inicio URNA x 1** // Na validação do acesso se estiver configurado é ativado o acionamento da urna mesmo com erro de leitura do cartão smartcard.
I [1] ExitAction //Ação de saída, 1=ligar, 2=desligar

```
I [4]   ActionTime          //Tempo do acionamento da saída em milissegundos
I [1]   EnterDependence     //Acionamento dependente de entrada, 1=sim, 0=não
T       Fim URNA x 1

T       Inicio SOLENOIDES  x 2
I [1]   ExitAction          //Ação de saída, 1=ligar, 0=desligar
                            Se MRA, "Solenóide 1" funciona como configuração de ação da
                            saída para TRANSISTOR.
I [4]   ActionTime          //Tempo do acionamento da saída em milissegundos
                            Se MRA, é o tempo de acionamamento em milissegundos para
                            TRANSISTOR (Solenóide 1).
I [1]   EnterDependence     //Acionamento dependente de entrada, 1=sim, 0=não
                            Não usado em MRA.
T       Fim SOLENOIDES  x 2

T       Inicio SAÍDASx 8
I [1]   ExitAction          //Ação de saída, 1=ligar, 2=desligar
                            Se MRA, "Saída 1" funciona como configuração de ação da
                            saída para RELÉ.
I [4]   ActionTime          //Tempo do acionamento da saída em milissegundos
                            Se MRA, é o tempo de acionamamento em milissegundos para
                            RELÉ (Saída 1).
I [1]   EnterDependence     //Acionamento dependente de entrada, 1=sim, 0=não
                            Não usado em MRA.
T       Fim SAÍDASx 8

T       Inicio PICTOGRAMA A DIREITA x 1
A [7]   ExitAction          //Ação de saída, estrutura texto (b1;b2;b3;b4),
                            b1 á b3 valores 0 ou 1 .Ex ("0;0;0;0")
                            b1 (1= tem  acionamento e 0= não tem);
                    '       b2 (1= liga seta anti horária e 0= desliga);
                            b3 (1=liga seta horária e 0= desliga)
                            b4 (1= liga X e   0= desligar X)
I [4]   ActionTime          //Tempo do acionamento da saída em  milissegundos
I [1]   EnterDependence     //Acionamento dependente de entrada, 1=sim, 0=não
T       Fim PICTOGRAMA A DIREITA x 1

T       Inicio PICTOGRAMA A ESQUERDA x 1
A [7]   ExitAction          //Ação de saída, estrutura texto (b1;b2;b3;b4),
                            b1 á b3 valores 0 ou 1 .Ex ("0;0;0;0")
                            b1 (1= tem  acionamento e 0= não tem);
                    '       b2 (1= liga seta anti horária e 0= desliga);
                            b3 (1=liga seta horária e 0= desliga)
                            b4 (1= liga X e   0= desligar X)
I [4]   ActionTime          //Tempo do acionamento da saída em  milissegundos
I [1]   EnterDependence     //Acionamento dependente de entrada, 1=sim, 0=não
T       Fim PICTOGRAMA A ESQUERDA x 1

T       Inicio CATRACA x 1  // Se configurar catraca não terá efeito as estrutruras:
                            Pictograma, Entradas 1 e 2 (monitora giro) e Solenóides 1 e 2
                            (trava sentido)
A [7]   ExitAction          //Ação de saída, estrutura texto (b1;b2;b3;b4),
```

b1 á b3 valores 0 ou 1 .Ex ("0;0;0;0")
"**1**;0;b3;b4" b1 (1=sentido horário);
"0;**1**;b3;b4" b2 (1=sentido anti-horário);
"**1**;**1**;b3;b4" b1,b2(1=ambos sentidos) ;
"b1;b2;**1**;b4" b3 (1=combinações de giros anteriores porém para CLIP);
"b1;b2;b3;**1**" b4 (1=combinações de giros anteriores porém para MCANet).

I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não
T **Fim CATRACA x 1**

T **Inicio DISPLAY x 1**
I [1] ExitAction //Ação de saída, valor 1=mostrar/ o resto não faz nada
I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não
T **Fim DISPLAY x 1**

S [16] DisplayPrimeiraLinha //Texto da primeira linha do display, prencher com zeros para não exibir texto
S [16] DisplaySegundaLinha //Texto da segunda linha do display, preencher com zeros para não exibir texto

T **Inicio SINALIZAÇÃO DE LEITORA x 1** //Utiliza esta configuração para leitoras externas do tipo Ativas que possuem sinalizadores de leds e buzzer
I [1] Action //Ação, Valores :
1=apenas buzzer,
2 =apenas led verde,
3=buzzer e led verde,
4=apenas led vermelho,
5=buzzer e led vermelho
I [4] BuzzerTime //Tempo de acionamento do buzzer em milissegundos
I [4] LedTime //Tempo de acionamento do led em milissegundos
I [1] EnterDependence //Acionamento depende de entrada, 1=sim, 0=não
T **Fim SINALIZAÇÃO DE LEITORA x 1**

T **Fim ACÕES DE ACESSO NEGADO x 1**

T **Inicio ACÕES PARA ACESSO COM REVISTA ALEATÓRIA x 1**
I [1] EnterDependence // Número da entrada (1,2,3 ou 4), dependente para o acionamento, utilizada ao invés de tempo para acionamento

T **Inicio PICTOGRAMA x 1**
A[7] ExitAction //Ação de saída, estrutura texto (b1;b2;b3;b4),
b1 (1= tem acionamento e 0= não tem);
b2 (1= liga seta anti horária e 0 = desliga);
b3 (1=liga seta horária e 0= desliga) e
b4 (1= ligaX e 0 = desligar X)
// Caso controladora MCANet
"1;0;0;1" = vermelho
"1;1;0;0" = verde

		"1;1;0;1" = verde e vermelho
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim,0=não
T	**Fim PICTOGRAMA x 1**	
T	**Inicio BUZZER x 1**	
I [1]	ExitAction	//Ação de saída, 1=ligar, 2=desligar
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim,0=não
T	**Fim BUZZER x 1**	
T	**Inicio URNA x 1**	// Na validação do acesso se estiver configurado Urna ele ativa o acionamento da urna mesmo com erro de leitura do cartão smartcard
I [1]	ExitAction	//Ação de saída, 1=ligar, 0=desligar
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim URNA x 1**	
T	**Inicio SOLENOIDES x 2**	
I [1]	ExitAction	//Ação de saída, 1=ligar, 0=desligar
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim SOLENOIDES x 2**	
T	**Inicio SAÍDAS x 8**	
I [1]	ExitAction	//Ação de saída, 1=ligar, 0=desligar
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim SAÍDAS x 8**	
T	**Inicio PICTOGRAMA A DIREITA x 1**	
A [7]	ExitAction	//Ação de saída, estrutura texto (b1;b2;b3;b4), formato "0;0;0;0" . b1 (1 = tem acionamento e 0 = não tem); b2 (1 = liga seta anti horária e 0 = desliga); b3 (1 =liga seta horária e 0 = desliga) e b4 (1 = liga X e 0 = desligar X)
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim PICTOGRAMA A DIREITA x 1**	
T	**Inicio PICTOGRAMA A ESQUERDA x 1**	
A [7]	ExitAction	//Ação de saída, estrutura texto (b1;b2;b3;b4), formato "0;0;0;0" . b1 (1 = tem acionamento e 0 = não tem); b2 (1 = liga seta anti horária e 0 = desliga); b3 (1 =liga seta horária e 0 = desliga) e b4 (1 = liga X e 0 = desligar X)
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim PICTOGRAMA A ESQUERDA x 1**	

T	**Inicio CATRACA x 1**	// Se configurar catraca não terá efeito as estruturas: Pictograma, Entradas 1 e 2 (monitora giro) e Solenóides 1 e 2 (trava sentido)
A [7]	ExitAction	//Ação de saída, estrutura texto (b1;b2;b3;b4), b1 á b3 valores 0 ou 1 .Ex ("0;0;0;0") "**1**;0;b3;b4" b1 (1=sentido horário); "0;**1**;b3;b4" b2 (1=sentido anti-horário); "**1**;**1**;b3;b4" b1,b2(1=ambos sentidos) ; "b1;b2;**1**;b4" b3 (1=combinações de giros anteriores porém para CLIP); "b1;b2;b3;**1**" b4 (1=combinações de giros anteriores porém para MCANet).
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim CATRACA x 1**	
T	**Inicio DISPLAY x 1**	
I [1]	ExitAction	//Ação de saída, valor 1=mostrar/ o resto não faz nada
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim DISPLAY x 1**	
S [16]	DisplayPrimeiraLinha	//Texto da primeira linha do display, prencher com zeros para não exibir texto
S [16]	DisplaySegundaLinha	//Texto da segunda linha do display, preencher com zeros para não exibir texto
T	**Inicio SINALIZAÇÃO DE LEITORA x 1**	//Utiliza esta configuração para leitoras externas do tipo Ativas que possuem sinalizadores de leds e buzzer

I [1]	Action	//Ação, Valores :
		1=apenas buzzer,
		2 =apenas led verde,
		3=buzzer e led verde,
		4=apenas led vermelho,
		5=buzzer e led vermelho

I [4]	BuzzerTime	//Tempo de acionamento do buzzer em milissegundos
I [4]	LedTime	//Tempo de acionamento do led em milissegundos
I [1]	EnterDependence	//Acionamento depende de entrada, 1=sim, 0=não
T	**Fim SINALIZAÇÃO DE LEITORA x 1**	
T	**Fim ACÕES PARA ACESSO COM REVISTA ALEATÓRIA x 1**	
I [4]	TimeouAntidupla	//Timeout da anti-dupla em milessegundos
T	**Fim LEITORASx 22**	
I[4]	TimeInterval	//Tempo de intervalo de inter-jornada em minutos
T	**Inicio ACIONAMENTOS DE EMERGÊNCIA x 1**	

```
I [1]   EnterDependence     // Número da entrada (1,2,3 ou 4), dependente para o
                            acionamento, utilizada ao invés de tempo para acionamento

T       Inicio PICTOGRAMA x 1
A [7]   ExitAction          //Ação de saída, estrutura texto (b1;b2;b3;b4),
                            //Caso controladora MCANet
                            "1;0;0;1" = vermelho
                            "1;1;0;0" = verde
                            "1;1;0;1" = verde e vermelho
I [4]   ActionTime          //Tempo do acionamento da saída em milissegundos
I [1]   EnterDependence     //Acionamento dependente de entrada, 1=sim,0=não
T       Fim PICTOGRAMA x 1

T       Inicio BUZZER x 1
I [1]   ExitAction          //Ação de saída, 1=ligar, 2=desligar
I [4]   ActionTime          //Tempo do acionamento da saída em milissegundos
I [1]   EnterDependence     //Acionamento dependente de entrada, 1=sim,0=não
T       Fim BUZZER x 1

T       Inicio URNA x 1      // Na validação do acesso se estiver configurado Urna ele ativa
                            o acionamento da urna mesmo com erro de leitura do cartão
                            smartcard
I [1]   ExitAction          //Ação de saída, 1=ligar, 2=desligar
I [4]   ActionTime          //Tempo do acionamento da saída em milissegundos
I [1]   EnterDependence     //Acionamento dependente de entrada, 1=sim, 0=não
T       Fim URNA x 1

T       Inicio SOLENOIDES  x 2
I [1]   ExitAction          //Ação de saída, 1=ligar, 2=desligar
I [4]   ActionTime          //Tempo do acionamento da saída em milissegundos
I [1]   EnterDependence     //Acionamento dependente de entrada, 1=sim, 0=não
T       Fim SOLENOIDES  x 2

T       Inicio SAÍDAS x 8
I [1]   ExitAction          //Ação de saída, 1=ligar, 2=desligar
I [4]   ActionTime          //Tempo do acionamento da saída em milissegundos
I [1]   EnterDependence     //Acionamento dependente de entrada, 1=sim, 0=não
T       Fim SAÍDAS x 8

T       Inicio PICTOGRAMA A DIREITA x 1
A [7]   ExitAction          //Ação de saída, estrutura texto (b1;b2;b3;b4),
                            formato "0;0;0;0" .
                            b1 (1 = tem  acionamento e 0 = não tem);
                            b2 (1 = liga seta anti horária e 0 = desliga);
                            b3 (1 =liga seta horária e 0 = desliga) e
                            b4 (1 = liga X e 0 = desligar X)
I [4]   ActionTime          //Tempo do acionamento da saída em  milissegundos
I [1]   EnterDependence     //Acionamento dependente de entrada, 1=sim, 0=não
T       Fim PICTOGRAMA A DIREITA x 1

T       Inicio PICTOGRAMA A ESQUERDA x 1
A [7]   ExitAction          //Ação de saída, estrutura texto (b1;b2;b3;b4),
```

		formato "0;0;0;0" . b1 (1 = tem acionamento e 0 = não tem); b2 (1 = liga seta anti horária e 0 = desliga); b3 (1 =liga seta horária e 0 = desliga) e b4 (1 = liga X e 0 = desligar X)
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim PICTOGRAMA A ESQUERDA x 1**	
T	**Inicio CATRACA x 1**	// Se configurar catraca não terá efeito as estruturas: Pictograma, Entradas 1 e 2 (monitora giro) e Solenóides 1 e 2 (trava sentido)
A [7]	ExitAction	//Ação de saída, estrutura texto (b1;b2;b3;b4), b1 á b3 valores 0 ou 1 .Ex ("0;0;0;0") "**1**;0;b3;b4" b1 (1=sentido horário); "0;**1**;b3;b4" b2 (1=sentido anti-horário); "**1**;**1**;b3;b4" b1,b2(1=ambos sentidos) ; "b1;b2;**1**;b4" b3 (1=combinações de giros anteriores porém para CLIP); "b1;b2;b3;**1**" b4 (1=combinações de giros anteriores porém para MCANet).
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim CATRACA x 1**	
T	**Inicio DISPLAY x 1**	
I [1]	ExitAction	//Ação de saída, valor 1=mostrar/ o resto não faz nada
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim DISPLAY x 1**	
S [16]	DisplayPrimeiraLinha	//Texto da primeira linha do display, prencher com zeros para não exibir texto
S [16]	DisplaySegundaLinha	//Texto da segunda linha do display, preencher com zeros para não exibir texto
T	**Inicio SINALIZAÇÃO DE LEITORA x 1**	//Utiliza esta configuração para leitoras externas do tipo Ativas que possuem sinalizadores de leds e buzzer

I [1]	Action	//Ação, Valores :
		1=apenas buzzer,
		2 =apenas led verde,
		3=buzzer e led verde,
		4=apenas led vermelho,
		5=buzzer e led vermelho

I [4]	BuzzerTime	//Tempo de acionamento do buzzer em milissegundos
I [4]	LedTime	//Tempo de acionamento do led em milissegundos
I [1]	EnterDependence	//Acionamento depende de entrada, 1=sim, 0=não
T	**Fim SINALIZAÇÃO DE LEITORA x 1**	
T	**Fim ACIONAMENTOS DE EMERGÊNCIA x 1**	

T	**Inicio ACIONAMENTOS DE ALARMES x 4**	
I [1]	EnterDependence	// Número da entrada (1,2,3 ou 4), dependente para o acionamento, utilizada ao invés de tempo para acionamento
T	**Inicio PICTOGRAMA x 1**	

A [7]	ExitAction	//Ação de saída, estrutura texto (b1;b2;b3;b4), formato "0;0;0;0" .
		b1 (1 = tem acionamento e 0 = não tem);
		b2 (1 = liga seta anti horária e 0 = desliga);
		b3 (1 =liga seta horária e 0 = desliga) e
		b4 (1 = liga X e 0 = desligar X)
		//Caso controladora MCANet
		"1;0;0;1" = vermelho
		"1;1;0;0" = verde
		"1;1;0;1" = verde e vermelho

I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim,0=não
T	**Fim PICTOGRAMA x 1**	
T	**Inicio BUZZER x 1**	
I [1]	ExitAction	//Ação de saída, 1=ligar, 0=desligar
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim,0=não
T	**Fim BUZZER x 1**	
T	**Inicio URNA x 1**	// Na validação do acesso se estiver configurado Urna ele ativa o acionamento da urna mesmo com erro de leitura do cartão smartcard
I [1]	ExitAction	//Ação de saída, 1=ligar, 0=desligar
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim URNA x 1**	
T	**Inicio SOLENOIDES x 2**	
I [1]	ExitAction	//Ação de saída, 1=ligar, 0=desligar
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim SOLENOIDES x 2**	
T	**Inicio SAÍDAS x 8**	
I [1]	ExitAction	//Ação de saída, 1=ligar, 0=desligar
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim SAÍDAS x 8**	
T	**Inicio PICTOGRAMA A DIREITA x 1**	
A [7]	ExitAction	//Ação de saída, estrutura texto (b1;b2;b3;b4), formato "0;0;0;0" . b1 (1 = tem acionamento e 0 = não tem); b2 (1 = liga seta anti horária e 0 = desliga); b3 (1 =liga seta horária e 0 = desliga) e b4 (1 = liga X e 0 = desligar X)

I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não
T **Fim PICTOGRAMA A DIREITA x 1**

T **Inicio PICTOGRAMA A ESQUERDA x 1**
A [7] ExitAction //Ação de saída, estrutura texto (b1;b2;b3;b4),
formato "0;0;0;0" .
b1 (1 = tem acionamento e 0 = não tem);
b2 (1 = liga seta anti horária e 0 = desliga);
b3 (1 =liga seta horária e 0 = desliga) e
b4 (1 = liga X e 0 = desligar X)
I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não
T **Fim PICTOGRAMA A ESQUERDA x 1**

T **Inicio CATRACA x 1** // Se configurar catraca não terá efeito as estrutruras:
//Pictograma, Entradas 1 e 2 (monitora giro) e Solenóides 1 e 2 (trava sentido)
A [7] ExitAction //Ação de saída, estrutura texto (b1;b2;b3;b4),
b1 á b3 valores 0 ou 1 .Ex ("0;0;0;0")
"**1**;0;b3;b4" b1 (1=sentido horário);
"0;**1**;b3;b4" b2 (1=sentido anti-horário);
"**1**;**1**;b3;b4" b1,b2(1=ambos sentidos) ;
"b1;b2;**1**;b4" b3 (1=combinações de giros anteriores porém para CLIP);
"b1;b2;b3;**1**" b4 (1=combinações de giros anteriores porém para MCANet).
I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não
T **Fim CATRACA x 1**

T **Inicio DISPLAY x 1**
I [1] ExitAction //Ação de saída, valor 1=mostrar/ o resto não faz nada
I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não
T **Fim DISPLAY x 1**

S [16] DisplayPrimeiraLinha //Texto da primeira linha do display, prencher com zeros para não exibir texto
S [16] DisplaySegundaLinha //Texto da segunda linha do display, preencher com zeros para não exibir texto

T **Inicio SINALIZAÇÃO DE LEITORA x 1** //Utiliza esta configuração para leitoras externas do tipo Ativas que possuem sinalizadores de leds e buzzer
I [1] Action //Ação, Valores :
1=apenas buzzer,
2 =apenas led verde,
3=buzzer e led verde,
4=apenas led vermelho,
5=buzzer e led vermelho
I [4] BuzzerTime //Tempo de acionamento do buzzer em milissegundos

I [4] LedTime //Tempo de acionamento do led em milissegundos
I [1] EnterDependence //Acionamento depende de entrada, 1=sim, 0=não
T **Fim SINALIZAÇÃO DE LEITORA x 1**

T **Fim ACIONAMENTOS DE ALARMES x 4**

T **Início MENSAGENS DE EVENTOS x 32** // Esta estrutura é utilizada para substituição das mensagens padrões do firmware.
I[3] CodEvento //Código do evento
I[4] ExibTime //Tempo de exibição da mensagem emmilissegundos
I [1] ConfExib1 //Configuração de exibição da linha 1 do display, 0=mensagem padrão, 1=texto personalizado, 2=data e hora, 3=número do cartão
S [16] PersText1 //Texto personalizado caso configuração linha1
I [1] ConfExib2 //Configuração de exibição da linha 2 do display, 0=mensagem padrão, 1=texto personalizado, 2=data e hora, 3=número do cartão
S [16] PersText2 //Texto personalizado caso configuração linha2
T **Fim MENSAGENS DE EVENTOS x 32**

I [1] AuthorizerTimeout //Timeout da lista do autorizador em milissegundos
I[1] SmartCardValidation //Tipo de validação smartcard
0=smart com mapa,
1=smart como proximidade
2=portaria 59/65 (controle de recinto alfandegado da receita federal) (ISPS Code USA)
I[4] EnclosureCode //Código do recinto BDCC
[1024] CertModBDCC //módulos do certificado BDCC
B [64] CertExpBDCC //Exponente do certificado BDCC

Retorno do envio de Comandos:
0 = Comando Enviado com Sucesso ou -1 = Erro // Resposta ->MyListener sessão 6.1

Exemplo C#:

Envio de Comando Assincrono:

```
ConfigFirmware obj = new ConfigFirmware();
        obj.CommStatus = 2;
        obj.DeviceID = 1;
        obj.SeqCmd = 125;

// Este comando contém muitos parâmetros que deverão todos ser preenchidos aqui na
// sequência conforme declarações destes parâmetros acima descritos.

{
        Console.WriteLine("Erro");
}
else
{
        Console.WriteLine("Comando enviado com sucesso – Aguarde retorno!");
}
```

Recepção de Resposta Assincrona:

```
  if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
      {
       String BuffMsg = "";
       if (args.Msg is ConfigFirmwareResp)
        {
         DFS.Ack(args.Msg.DeviceID);
      BuffMsg = "\nRetorno Tipo  " +
(CommandReturnType)((CommandReturn)args.Msg).CommandId +
  "\nDeviceID" + ((ConfigFirmwareResp )args.Msg).DeviceId.ToString() +
  "\nSequencia CMD " + ((ConfigFirmwareResp)args.Msg).CmdSeq.ToString() +
  "\nCodigo de ret " + ((ConfigFirmwareResp)args.Msg).ReturnCode.ToString() +
  "\nData Ger CMD" +
((ConfigFirmwareResp)args.Msg).CmdGenerationTimestamp.ToString() +
  "\nData Exe CMD " +
((ConfigFirmwareResp)args.Msg).CmdExecutionTimestamp.ToString() +
  "\nTam dos Dados " + ((ConfigFirmwareResp)args.Msg).PayloadSize.ToString();

Console.WriteLine(BuffMsg );
 }
}
```

9.6.2. Solicitar Backup dos Acessos – BackupAccess()

Descrição:
Esta função solicita os eventos da área de backup dos acessos registrados

Função:
Int Execute(obj BackupAcess);

Parametros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando
D	EventInitDateTime	// Data e Hora Inicial “dd/mm/aaaahh:mm:ss”
D	EventEndDateTime	// Data e Hora final “dd/mm/aaaa hh:mm:ss”

Retorno do envio de Comandos:
0 = Comando Enviado com Successo ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:

I [4]	TotEvent	//Quantidade de Eventos
T	Eventst[]	// Lista de eventos de alarmes baseado na quantidade
I [1]	IdentificationType	// Tipo de identificador , 1 = cartão; 2 = id da pessoa.
B [23]	IdentificationData	// Identificador do cartão ou da pessoa, conforme campo acima.Quando cartão, os primeiros 5 bytes representam o identificador, e o restante dos bytes devem ser lidos e descartados, virão zerados
I[1]	EventType	//Tipo do evento 0=offline,1=online
I[1]	SummerTime	//Horário de verão, 0 = não,1 = sim
I	GMT1	GMT (-3) = -180
I [1]	AcessDirection	//Direção do acesso,1 = enrtrada, 2 = saída

I ReaderID // Código da leitora

Exemplo C#:
Envio de Comando:

```
BackupAccess obj = new BackupAccess();
            obj.CommStatus = 2;
            obj.DeviceID = 1;
            obj.SeqCmd = 126;
            obj.EventInitDateTime = "01/01/2012 00:00:00";
            obj.EventEndDateTime = "01/10/2012 00:00:00";
            if (DFS.Execute(obj) < 0) {
}
else
            {
Console.WriteLine(" Comando enviado com sucesso");
            }
```

Recepção de Resposta Assincrona:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
{
String BuffMsg = "";
if (args.Msg is AccessBackupResp)
{
DFS.Ack(args.Msg.DeviceID);
string  BuffMsg ="Retorno de Comandos" +
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID  " + ((AccessBackupResp)args.Msg).DeviceId.ToString()  +
"\nSequencia CMD " + ((AccessBackupResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((AccessBackupResp)args.Msg).ReturnCode.ToString()  +
"\nData Ger CMD " +     ((AccessBackupResp)args.Msg).CmdGenerationTimestamp.ToString() +
 "\nData Exe CMD  " +   ((AccessBackupResp)args.Msg).CmdExecutionTimestamp.ToString()  +
"\nTam dos Dados " + ((AccessBackupResp)args.Msg).PayloadSize.ToString();

BuffMsg += "\nTotal Access = " + ((AccessBackupResp)args.Msg).TotEvent.ToString() + "\n\n" +
          "-----------------------------------------------------------------\n";
          for (int i = 0, n = 1; i < (int)((AccessBackupResp)args.Msg).TotEvent; i++, n++)
          {
BuffMsg += "Evento nº " + n.ToString() +
"\nId Evento  = " + ((AccessBackupResp)args.Msg).Events[i].EventID.ToString() +
"\nDate    = " + ((AccessBackupResp)args.Msg).Events[i].Timestamp.ToString() +
"\nDireção = " + ((AccessBackupResp)args.Msg).Events[i].AcessDirection.ToString() +
  "\nVerão   = " + ((AccessBackupResp)args.Msg).Events[i].SummerTime.ToString() +
"\ntipo    = " + ((AccessBackupResp)args.Msg).Events[i].IdentificationType.ToString() +
"\nid      = " +
BitConverter.ToString(((AccessBackupResp)args.Msg).Events[i].IdentificationData).Replace("-",
"") +
  "\nGmt     = " + ((AccessBackupResp)args.Msg).Events[i].GMT1.ToString() +
  "\nReader  = " + ((AccessBackupResp)args.Msg).Events[i].ReaderID.ToString() +
  "\nEsp Param = " + ((AccessBackupResp)args.Msg).Events[i].SpecificParameters.ToString() + "
\n " + "\n";
          }
Console.WriteLine(BuffMsg );
}
}
```

9.6.3. Solicitar Backup dos Alarmes –BackupAlarm()

Descrição:
Esta função obtem o backup da copia de segurança dos alarmes

Função:
Int Execute(BackupAlarm);

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando
D	EventInitDateTime	// Data Hora Inicial no formato dd/mm/aaaahh:mm:ss
D	EventEndDateTime	// DataHora final no formato dd/mm/aaaa hh:mm:ss

Retorno do envio de Comandos:
0 = Comando Enviado com Sucesso ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:

I [4]	`TotEvent`	//Quantidade de Eventos
T	`Eventst[]`	// Lista de eventos de alarmes baseado na quantidade
I [1]	IdentificationType	// Tipo de identificador , 1 = cartão; 2 = id da pessoa.
B [23]	IdentificationData	// Identificador do cartão ou da pessoa, conforme campo acima.Quando cartão, os primeiros 5 bytes representam o identificador, e o restante dos bytes devem ser lidos e descartados, virão zerados
I	SummerTime	//Horário de verão, 0 = não,1 = sim
I	GMT1	
I	AcessDirection	//Direção do acesso,1 = enrtrada, 2 = saída
I	ReaderID	// Código da leitora
		- Controle de Memória para Eventos Percentual de memória livre (1 byte)

Exemplo C#:
Envio de Comando:

```
BackupAlarm obj = new BackupAlarm();
                    obj.CommStatus = 2;
                    obj.DeviceID = 1;
                    obj.SeqCmd = 127;
                    obj.EventInitDateTime = “01/01/2012 00:00:00”;
                    obj.EventEndDateTime = “01/03/2012 00:00:00”;
 if (Execute (obj) < 0)
{
                    Console.WriteLine(“Erro");}else{
                    Console.WriteLine(“Comando enviado com sucesso – Aguarde retorno!”);
}
```

Recepção de Resposta Assincrona:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
                {
if(args.Msg.GetType().ToString().CompareTo("DigiconFrameworkServer.Objects.Messag
eObjects.AsyncObjects.AlarmBackupResp") == 0)
{
DFS.Ack(args.Msg.DeviceID);
       String acc = "-------------- AlarmBackupResp Backup-------------- " +
       "\n\nDeviceID " + ((AlarmBackupResp)args.Msg).DeviceID.ToString();
                            acc += "Total Alarm = " +
((AlarmBackupResp)args.Msg).TotEvent.ToString() + "\n";
for (int i = 0; i < (int)((AlarmBackupResp)args.Msg).TotEvent; i++)
                            {
 acc += "Id Evento ="+ ((AlarmBackupResp)args.Msg).Events[i].EventID.ToString() +
 "\nDate    =" + ((AlarmBackupResp)args.Msg).Events[i].Timestamp.ToString() +
"\nDireção =" + ((AlarmBackupResp)args.Msg).Events[i].AcessDirection.ToString() +
"\nVerão   =" + ((AlarmBackupResp)args.Msg).Events[i].SummerTime.ToString() +
"\ntipo ="+ ((AlarmBackupResp)args.Msg).Events[i].IdentificationType.ToString() +
"\nid =" +
BitConverter.ToString(((AlarmBackupResp)args.Msg).Events[i].IdentificationData).R
eplace("-", "") +
"\nGmt     =" + ((AlarmBackupResp)args.Msg).Events[i].GMT1.ToString() +
 "\nReader  =" + ((AlarmBackupResp)args.Msg).Events[i].ReaderID.ToString() +
"\nEsp Param =" +
((AlarmBackupResp)args.Msg).Events[i].SpecificParameters.ToString() + " \n\n ";
       }
Console.WriteLine(acc);
       }
```

9.6.4. Bloquear uma leitora – BlockDevice()

Descrição:
Esta função bloqueia um determinado dispositivo ou apenas uma leitora.

Função:
Int Execute(BlockDevice);

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando
A [43]	Readers	//Array das leitoras a serem bloqueadas, “0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0”, estrutura todas zerada bloqueia o dispositivo como um todo.

Retorno do envio de Comandos:
0 = Comando Enviado com Sucesso ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:
Não há dados de retorno

Exemplo C#:
Envio do Comando:
BlockDevice obj = new BlockDevice ();

```
                obj.CommStatus = 2;
                obj.DeviceID = 1;
                obj.SeqCmd = 128;
                obj.Readers = "1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1";
if (DFS.Execute(obj) < 0){
/*erro*/
}
else{
                Console.WriteLine("Comando enviado com sucesso – Aguarde retorno!");
}
```

Recepção de Resposta Assincrona:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
  {
  String BuffMsg = "";
if (args.Msg is BlockDeviceResp)
   {
   DFS.Ack(args.Msg.DeviceID);
   BuffMsg =
   "\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
   "\nDeviceID  + ((BlockDeviceResp)args.Msg).DeviceId.ToString() +
   "\nSequencia CMD " + ((BlockDeviceResp)args.Msg).CmdSeq.ToString() +
   "\nCodigo de ret " + ((BlockDeviceResp)args.Msg).ReturnCode.ToString() +
   "\nData Ger CMD  " +
       ((BlockDeviceResp)args.Msg).CmdGenerationTimestamp.ToString() +
  "\nData Exe CMD  " +
       ((BlockDeviceResp)args.Msg).CmdExecutionTimestamp.ToString()  +
 "\nTam dos Dados " + ((BlockDeviceResp)args.Msg).PayloadSize.ToString();

Console.WriteLine(BuffMsg );
                    }
}
```

9.6.5. Desbloquear uma Leitura – UnblockDevice()

Descrição:
Esta função desbloqueia um determinado dispositivo

Função:
Int Execute(UnBlockDevice);

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando
A [43]	Readers	//Array das leitoras a serem bloqueadas, "0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0;0", estrutura todas zerada desbloqueia o dispositivo como um todo.

Retorno do envio de Comandos:
0 = Comando Enviado com Successo ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:
Não há dados de retorno

Exemplo C#:
Envio do Comando:

```
UnBlockDevice obj = new UnBlockDevice ();
                obj.CommStatus = 2;
                obj.DeviceID = 1;
                obj.SeqCmd = 128;
                obj.Readers = "1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1;1";
        if (DFS.Execute(obj) < 0){
        /*erro*/
        }else{

                Console.WriteLine("Comando enviado com sucesso!");
}
```

Recepção de Resposta Assincrona:

```
  if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
  {
  String BuffMsg = "";
  if (args.Msg is UnblockDeviceResp)
 {
   DFS.Ack(args.Msg.DeviceID);
BuffMsg =
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID" + ((UnblockDeviceResp)args.Msg).DeviceId.ToString() +
"\nSequencia CMD " + ((UnblockDeviceResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((UnblockDeviceResp)args.Msg).ReturnCode.ToString() +
 "\nData Ger CMD  " +
       (UnblockDeviceResp)args.Msg).CmdGenerationTimestamp.ToString() +
"\nData Exe CMD  " +
       ((UnblockDeviceResp)args.Msg).CmdExecutionTimestamp.ToString()  +
"\nTam dos Dados " + ((UnblockDeviceResp)args.Msg).PayloadSize.ToString();

Console.WriteLine(BuffMsg );
                    }
}
```

9.6.6. Atualizar Firmware – UpdateFirmware()

Descrição:
Esta função atualiza o firmware de determinado dispositivo.
No caso da controladora MCANet, é possível atualizar os os fontes utilizados no display, o processo é o mesmo, basta informar o path do pacote "fontesTTF-digicon.tar.gz" .

Função:
Int Execute(UpdateFirmware).

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando

S [x] Firmware // Path onde se encontra o firmware com nome

Retorno do envio de Comandos:
0 = Comando Enviado com Sucesso ou -1 = Erro //

Dados da Resposta do comando:
Não há dados de retorno

Observação:

Para a atualização da controladora **MCA** com versões anteriores a 2.1, é preciso enviar o pacote firmware-digicon-mca-base.vx.x.x.tar.gz e após a reinicialização do equipamento, enviar o pacote com a versão 2.1 ou superior.
A atualização da controladora **MCANet** basta enviar o pacote firmware-digicon.vx.x.x.tar.gz .

Exemplo C#:
Envio do Comando:

```
UpdateFirmware obj = new UpdateFirmware ();
                    obj.CommStatus = 2;
                    obj.DeviceID = 1;
                    obj.SeqCmd = 128;
                    obj.Firmware = "C:\ firmware-digicon.vX.X.X.tar.gz ";
        if (DFS.Execute(obj) < 0) {

 } else
        {
                    Console.WriteLine(" Comando enviado com sucesso");
}
```

9.6.7. Ativar_emergência – EnableEmergency()

Descrição:
Esta função ativa o dispositivo para modo emergência.

Função:
Int Execute(EnableEmergency);

Parâmetros:
I [1] CommStatus // Estado da comunicação,1=Offline, 2=Online
I [9] DeviceID // ID do dispositivo para onde deverá ser enviado o comando
I [9] SeqCmd // Sequencial do comando

Retorno do envio de Comandos:
0 = Comando Enviado com Sucesso ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:
Não há dados de retorno

Exemplo C#:
Envio do Comando:

```
EnableEmergency obj = new EnableEmergency ();
                obj.CommStatus = 2;
                obj.DeviceID = 1;
                obj.SeqCmd = 128;
if (DFS.Execute(obj) < 0){

 /*erro*/}

else{
                Console.WriteLine("Comando enviado com sucesso!");

}
```

Recepção de Resposta Assincrona:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
  {
    String BuffMsg = "";
    if (args.Msg is EnableEmergencyResp)
{
  DFS.Ack(args.Msg.DeviceID);
string  BuffMsg =
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID      " + ((EnableEmergencyResp)args.Msg).DeviceId.ToString()   +
"\nSequencia CMD " + ((EnableEmergencyResp)args.Msg).CmdSeq.ToString()  +
"\nCodigo de ret " + ((EnableEmergencyResp)args.Msg).ReturnCode.ToString() +
 "\nData Ger CMD  " +
((EnableEmergencyResp)args.Msg).CmdGenerationTimestamp.ToString() +
"\nData Exe CMD  " +
((EnableEmergencyResp)args.Msg).CmdExecutionTimestamp.ToString()  +
"\nTam dos Dados " + ((EnableEmergencyResp)args.Msg).PayloadSize.ToString();

Console.WriteLine(BuffMsg );
     }
}
```

9.6.8. Desativar Emergência – DisableEmergency()

Descrição:
Esta função desativa o modo de emergência do dispositivo

Função:
Int Execute(DisableEmergency);

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando

Retorno do envio de Comandos:
0 = Comando Enviado com Successo ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:

Não há dados de retorno

Exemplo C#:
Envio do Comando:

```
DisableEmergency obj = new DisableEmergency ();
                obj.CommStatus = 2;
                obj.DeviceID = 1;
                obj.SeqCmd = 128;
if (DFS.Execute(obj) < 0){
 /*erro*/
}else{
                Console.WriteLine("Comando enviado com sucesso");
}
```

Recepção de Resposta Assincrona:

```
 if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
  {
  String BuffMsg = "";
if (args.Msg is DisableEmergencyResp)
   {
DFS.Ack(args.Msg.DeviceID);
BuffMsg =
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID  " + ((DisableEmergencyResp)args.Msg).DeviceId.ToString() +
"\nSequencia CMD " + ((DisableEmergencyResp)args.Msg).CmdSeq.ToString()  +
"\nCodigo de ret " + ((DisableEmergencyResp)args.Msg).ReturnCode.ToString() +
"\nData Ger CMD  " +
      ((DisableEmergencyResp)args.Msg).CmdGenerationTimestamp.ToString() +
"\nData Exe CMD  " +
      ((DisableEmergencyResp)args.Msg).CmdExecutionTimestamp.ToString()  +
"\nTam dos Dados " + ((DisableEmergencyResp)args.Msg).PayloadSize.ToString();

Console.WriteLine(BuffMsg );
                    }
}
```

9.6.9. Ajustar Horario de Verão – UpdateSummerTime()

Descrição:
Esta função atualiza o horário de verão de um determinado dispositivo.

Função:
Int Execute(UpdateSummerTime);

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando
D	IniSummerTime	// Data/Hora Inicio Horário de Verão
D	EndSummerTime	// Data/Hora Fim Horário de Verão

Retorno do envio de Comandos:
0 = Comando Enviado com Successo ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:
Não há dados de retorno

Exemplo C#:
Envio do Comando:

```
UpdateSummerTime obj = new UpdateSummerTime ();
               obj.CommStatus = 2;
               obj.DeviceID = 1;
               obj.SeqCmd = 128;
               obj.InitSummerTime = "01/04/2012";
               obj.EndSummerTime = "30/07/2012";
        if (DFS.Execute(obj) < 0) {
}
else
        {
              Console.WriteLine (" Comando enviado com sucesso");
        }
```

Recepção de Resposta Assincrona:

```
 if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
 {
 String BuffMsg = "";

if (args.Msg is UpdateSummerTimeResp)
 {
 DFS.Ack(args.Msg.DeviceID);
BuffMsg =
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID      " + ((UpdateSummerTimeResp)args.Msg).DeviceId.ToString() +
"\nSequencia CMD " + ((UpdateSummerTimeResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((UpdateSummerTimeResp)args.Msg).ReturnCode.ToString() +
"\nData Ger CMD  " +
       ((UpdateSummerTimeResp)args.Msg).CmdGenerationTimestamp.ToString() +
"\nData Exe CMD  " +
((UpdateSummerTimeRespp)args.Msg).CmdExecutionTimestamp.ToString()  +
"\nTam dos Dados " + ((UpdateSummerTimeResp)args.Msg).PayloadSize.ToString();

Console.WriteLine(BuffMsg );
                    }
}
```

9.6.10. Ligar Saida Digital – EnableDigitalOut()

Descrição:
Esta função ativa a saída digital de um determinado dispositivo.

Função:
IntExecute(EnableDigitalOut);

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando

B [1] Code // Saída Digital de 1 até 8 a ser ligada
I [4] Time // Duração em milisegundos, 0=constante

Retorno do envio de Comandos:
0 = Comando Enviado com Sucesso ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:
Não há dados de retorno

Exemplo C#:
Envio do Comando:

```
EnableDigitalOut obj = new EnableDigitalOut();
obj.CommStatus = 2;
            obj.DeviceID =1;
            obj.SeqCmd = 128;
            obj.Code = (byte)i;
            obj.Time = 100000;
            if (DFS.Execute(obj) < 0)
{ /*erro*/}
            else
            {
          Console.WriteLine("Comando enviado com sucesso");
                              }
```

Recepção de Resposta Assincrona:

```
  if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
 {
String BuffMsg = "";
 if (args.Msg is EnableDigitalOutResp)
{
  DFS.Ack(args.Msg.DeviceID);
BuffMsg =
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID      " + ((EnableDigitalOutResp)args.Msg).DeviceId.ToString() +
"\nSequencia CMD " + ((EnableDigitalOutResp)args.Msg).CmdSeq.ToString()  +
"\nCodigo de ret " + ((EnableDigitalOutResp)args.Msg).ReturnCode.ToString() +
"\nData Ger CMD  " +
       ((EnableDigitalOutResp)args.Msg).CmdGenerationTimestamp.ToString() +
"\nData Exe CMD  " +
       ((EnableDigitalOutResp)args.Msg).CmdExecutionTimestamp.ToString()  +
 "\nTam dos Dados " + ((EnableDigitalOutResp)args.Msg).PayloadSize.ToString();

Console.WriteLine(BuffMsg );
      }
}
```

9.6.11. Desligar Saida Digital – DisableDigitalOut()

Descrição:
Esta função desativa a saída digital de um determinado dispositivo.

Função:
Int Execute(DisableDigitalOut);

Parâmetros:

I [1] CommStatus // Estado da comunicação,1=Offline, 2=Online
I [9] DeviceID // ID do dispositivo para onde deverá ser enviado o comando
I [9] SeqCmd // Sequencial do comando
B [1] Code // Saída Digital de 1 até 8 a ser desligada

Retorno do envio de Comandos:
0 = Comando Enviado com Sucesso ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:
Não há dados de retorno

Exemplo C#:
Envio do Comando:

```
DisableDigitalOut obj = new DisableDigitalOut();
obj.CommStatus = 2;
obj.DeviceID = 1;
obj.SeqCmd = 128;
obj.Code= 5;    //desligar saída digital número 5
if (DFS.Execute(obj) < 0){
/*erro*/}
else{

Console.WriteLine("Comando enviado com sucesso");

}
```

Recepção de Resposta Assincrona:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
  {
String BuffMsg = "";
 if (args.Msg is DisableDigitalOutResp)
    {
        DFS.Ack(args.Msg.DeviceID);
string  BuffMsg = "Retorno de Comandos" +
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID  " + ((DisableDigitalOutResp)args.Msg).DeviceId.ToString()  +
"\nSequencia CMD " + ((DisableDigitalOutResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((DisableDigitalOutResp)args.Msg).ReturnCode.ToString()  +
"\nData Ger CMD " +
      ((DisableDigitalOutResp)args.Msg).CmdGenerationTimestamp.ToString() +
 "\nData Exe CMD  " +
      ((DisableDigitalOutResp)args.Msg).CmdExecutionTimestamp.ToString()  +
"\nTam dos Dados " + ((DisableDigitalOutResp)args.Msg).PayloadSize.ToString();

Console.WriteLine(BuffMsg );
     }
}
```

9.6.12. Obter Estado das Saídas Digitais – DigitalOutStatus()

Descrição:
Esta função solicita o status da saida digital de um determinado dispositivo.

Função:
Int Execute(DigitalOutStatus);

Parâmetros:
I [1] CommStatus // Estado da comunicação,1=Offline, 2=Online
I [9] DeviceID // ID do dispositivo para onde deverá ser enviado o comando
I [9] SeqCmd // Sequencial do comando

Retorno do envio de Comandos:
0 = Comando Enviado com Successo ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:
A [13] DigitalOutputStatus // 13 bytes estado das saídas digitais,0=desligada, 1=ligada

Exemplo C#:
Envio do Comando:

```
DigitalOutStatus obj = new DigitalOutStatus();
obj.CommStatus = 2;
obj.DeviceID = 1;
obj.SeqCmd = 128;
if (DFS.Execute(obj) < 0){ /*erro*/}
else{Console.WriteLine("Comando enviado com sucesso – Aguarde retorno!");
}
```

Recepção de Resposta Assincrona:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
{
String BuffMsg = "";
if (args.Msg is DigitalOutStatusResp)
{
DFS.Ack(args.Msg.DeviceID);
string  BuffMsg ="Retorno de Comandos" +
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID  " + ((DigitalOutStatusResp)args.Msg).DeviceId.ToString()  +
"\nSequencia CMD " + ((DigitalOutStatusResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((DigitalOutStatusResp)args.Msg).ReturnCode.ToString()  +
"\nData Ger CMD"+
       ((DigitalOutStatusResp)args.Msg).CmdGenerationTimestamp.ToString() +
 "\nData Exe CMD  " +
       ((DigitalOutStatusResp)args.Msg).CmdExecutionTimestamp.ToString()  +
"\nTam dos Dados " + ((DigitalOutStatusResp)args.Msg).PayloadSize.ToString();
for (int i = 0; i < ((DigitalOutStatusResp)args.Msg).OutStatus.Length; i++)
    {
    BuffMsg +=  ((DigitalOutStatusResp)args.Msg).OutStatus[i].ToString();
    }
    Console.WriteLine(BuffMsg );
      }
    }
```

9.6.13. Obter Estado das Entradas Digitais – DigitalInStatus()

Descrição:

Esta função solicita o status da entrada digital.

Função:
Int Execute(DigistalInStatus);

Parâmetros:
I [1] CommStatus // Estado da comunicação,1=Offline, 2=Online
I [4] DeviceID // ID do dispositivo onde se deseja configurar o firmware
I [4] SeqCmd // A aplicação DEVE controlar a sequencia dos comandos

Retorno do envio de Comandos:
0 = Comando Enviado com Successo ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:
A [7] DigitalInStatus // Array com 4 bytes estado da entrada digital,0=desligada, 1=ligada

Exemplo C#:
Envio do Comando:

```
DigitalInStatus obj = new DigitalInStatus();
                    obj.CommStatus = 2;
                    obj.DeviceID = 1;
                    obj.SeqCmd = 128;
if (DFS.Execute(obj) < 0){
/*erro*/}
else{
                    Console.WriteLine("Comando enviado com sucesso – Aguarde retorno!");
}
```

Recepção de Resposta Assincrona:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
{
String BuffMsg = "";
if (args.Msg is DigitalInStatusResp)
{
DFS.Ack(args.Msg.DeviceID);
string  BuffMsg ="Retorno de Comandos" +
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID  " + ((DigitalInStatusResp)args.Msg).DeviceId.ToString()  +
"\nSequencia CMD " + ((DigitalInStatusResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((DigitalInStatusResp)args.Msg).ReturnCode.ToString()  +
"\nData Ger CMD " +     ((DigitalInStatusResp)args.Msg).CmdGenerationTimestamp.ToString() +
 "\nData Exe CMD  " +   ((DigitalInStatusResp)args.Msg).CmdExecutionTimestamp.ToString()  +
"\nTam dos Dados " + ((DigitalInStatusResp)args.Msg).PayloadSize.ToString();

BuffMsg += "\nInputs ";
for (int i = 0; i < ((DigitalInStatusResp)args.Msg).InStatus.Length; i++)
{
        BuffMsg += ((DigitalInStatusResp)args.Msg).InStatus[i].ToString();
}
    Console.WriteLine(BuffMsg );
```

```
}
}
```

9.6.14. Carga do Mapa SmartCard– LoadSmartMap()

Descrição:
Esta função carrega o smartmap

Função:
Int Execute(LoadSmartMap, mapa de dados do Smart Card);

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando
B [1024]	SmartMap	// 1024 bytes com a representação do mapa SmartCard

Retorno do envio de Comandos:
0 = Comando Enviado com Sucesso ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:
Não há dados de retorno

Exemplo C#:
Envio do Comando:

```
LoadSmartMap obj = new LoadSmartMap();
            obj.CommStatus = 2;
            obj.DeviceID =
            obj.SeqCmd = 123;
            obj.Map = new Byte[1024];

            if (DFS.Execute(obj) < 0) { }
            else
            {
              Console.WriteLine(" Comando enviado com sucesso");
            }
```

Recepção de Resposta Assincrona:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
{
String BuffMsg = "";
if (args.Msg is LoadSmartMapResp)
{
DFS.Ack(args.Msg.DeviceID);
string  BuffMsg ="Retorno de Comandos" +
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID  " + ((LoadSmartMapResp)args.Msg).DeviceId.ToString()  +
"\nSequencia CMD " + ((LoadSmartMapResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((LoadSmartMapResp)args.Msg).ReturnCode.ToString()  +
"\nData Ger CMD " +    ((LoadSmartMapResp)args.Msg).CmdGenerationTimestamp.ToString()
+
 "\nData Exe CMD  " +   ((LoadSmartMapResp)args.Msg).CmdExecutionTimestamp.ToString()  +
```

```
"\nTam dos Dados " + ((LoadSmartMapResp)args.Msg).PayloadSize.ToString();

Console.WriteLine(BuffMsg );
}
}
```

9.6.15. Processos Automáticos - LoadAutomaticProcess()

Descrição:
Esta função carrega um processo automático.

Função:
Int Execute(LoadAutomaticProccess);

Parâmetros:

```
I [1]   CommStatus          // Estado da comunicação,1=Offline, 2=Online
I [9]   DeviceID            // ID do dispositivo para onde deverá ser enviado o comando
I [9]   SeqCmd              // Sequencial do comando
I [1]   Type                //Tipo da lista, 1=carga, 2=exclusão,3=status
I [5]   Code                // Codigo do processo Automático
        // Tipo = 1, Carga, deve ser informado bloco de processos
                automáticos.
                            // Tipo = 2, Exclusão, apenas o código do processo.
                            // Tipo = 3, Status, não informar nada o retorno será a lista de
                                    processos do dispositivo.

T       Inicio PROCESSOS AUTOMATICOS x 1// Estrutura válida para Tipo = 1 (Carga)
D       DateLastExec        // Data/Hora ultima execução
D       DateInitExec        // Data/Hora do Início da execução
D       DateEndExec         // Data/Hora do Final da execução
I [1]   Periodicity         // Período 1=uma vez, 2=Minuto, 5=Semana, 5=Mês
I [1]   Frequency           //Ocorrência dentro da Periodicidade, exemplo Mensale a Taxa
                                    = 2, rodará duas vezes no mês
A [8]   DaysOfTheWeek       // Estrutura de bytes “0;0;0;0;0;0;0;0”, representando os dias
                                    da semana para executar, iniciando no segundo byte, o
                                    primeiro sempre é zerado. Por exemplo: 00110011,
                                    executará domingo, segunda-feira, quinta-feira e sexta-
                                    feira.
I [1]   ExecHolidays        //Executa nos feriado, 1=sim 0=não

T       Inicio ACIONAMENTOS x 1
I [1]   EnterDependence     // Número da entrada (1,2,3 ou 4), dependente para o
                                    acionamento, utilizada ao invés de tempo para
                                    acionamento
T       Inicio PICTOGRAMA x 1
A [7]   ExitAction          //Ação de saída, estrutura texto (b1;b2;b3;b4), b1 (1= tem
                                    acionamento e 0= não tem); b2 (1= liga seta anti
                                    horária e 0 = desliga); b3 (1=liga seta horária e 0=
                                    desliga) e b4 (1= liga X e 2= desligar X)
                            //Caso MCANet :
                                    "1;0;0;1" = vermelho
```

"1;1;0;0" = verde
"1;1;0;1" = verde e vermelho

I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim,0=não
T **Fim PICTOGRAMA x 1**

T **Inicio BUZZER x 1**
I [1] ExitAction //Ação de saída, 1=ligar, 2=desligar
I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim,0=não
T **Fim BUZZER x 1**

T **Inicio URNA x 1** // Na validação do acesso se estiver configurado Urna ele ativa o acionamento da urna mesmo com erro de leitura do cartão smartcard
I [1] ExitAction //Ação de saída, 1=ligar, 2=desligar
I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não
T **Fim URNA x 1**

T **Inicio SOLENOIDES x 2**
I [1] ExitAction //Ação de saída, 1=ligar, 2=desligar
I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não
T **Fim SOLENOIDES x 2**

T **Inicio SAÍDAS x 8**
I [1] ExitAction //Ação de saída, 1=ligar, 2=desligar
I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não
T **Fim SAÍDAS x 8**

T **Inicio PICTOGRAMA A DIREITA x 1**
A [7] ExitAction //Ação de saída, estrutura texto (b1;b2;b3;b4), b1 (1= tem acionamento e 0= não tem); b2 (1= liga seta anti horária e 0= desliga); b3 (1=liga seta horária e 0= desliga) e b4 (1= liga X e 2= desligar X)
I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não
T **Fim PICTOGRAMA A DIREITA x 1**

T **Inicio PICTOGRAMA A ESQUERDA x 1**
A [7] ExitAction //Ação de saída, estrutura texto (b1;b2;b3;b4), b1 (1= tem acionamento e 0= não tem); b2 (1= liga seta anti horária e 0= desliga); b3 (1=liga seta horária e 0= desliga) e b4 (1= liga X e 2= desligar X)
I [4] ActionTime //Tempo do acionamento da saída em milissegundos
I [1] EnterDependence //Acionamento dependente de entrada, 1=sim, 0=não
T **Fim PICTOGRAMA A ESQUERDA x 1**

T	**Inicio CATRACA x 1**	// Se configurar catraca não terá efeito as estruturas: Pictograma, Entradas 1 e 2 (monitora giro) e Solenóides 1 e 2 (trava sentido)
A [7]	ExitAction	//Ação de saída, estrutura texto (b1;b2;b3;b4), b1 á b3 valores 0 ou 1 .Ex ("0;0;0;0") "**1**;0;b3;b4" b1 (1=sentido horário); "0;**1**;b3;b4" b2 (1=sentido anti-horário); "**1**;**1**;b3;b4" b1,b2(1=ambos sentidos) ; "b1;b2;**1**;b4" b3 (1=combinações de giros anteriores porém para CLIP); "b1;b2;b3;**1**" b4 (1=combinações de giros anteriores porém para MCANet).
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim CATRACA x 1**	
T	**Inicio DISPLAY x 1**	
I [1]	ExitAction	//Ação de saída, valor 1=mostrar/ o resto não faz nada
I [4]	ActionTime	//Tempo do acionamento da saída em milissegundos
I [1]	EnterDependence	//Acionamento dependente de entrada, 1=sim, 0=não
T	**Fim DISPLAY x 1**	
S [16]	DisplayPrimeiraLinha	//Texto da primeira linha do display, prencher com zeros para não exibir texto
S [16]	DisplaySegundaLinha	//Texto da segunda linha do display, preencher com zeros para não exibir texto
T	**Fim ACIONAMENTOS x 1**	
T	**Fim PROCESSOS AUTOMATICOS x 1**	

Retorno do envio de Comandos:
0 = Comando Enviado com Sucesso ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:

I [5]	`TotProcess`	// Quantidade de processos
I [5]	`ProcessCode`	// Código do processo automático
D	`ExeDateTime`	// Data da última execução dd/mm/aaaa hh:mm:SS

Exemplo C#:
Envio do Comando:

```
AutomaticProcess autoProc = new AutomaticProcess();
            autoProc.CommStatus = 2;
            autoProc.DeviceID = 1;
            autoProc.SeqCmd = 2;
            // buzzer
            autoProc.Process.Actuates.Buzzer.ExitAction = 1;
            autoProc.Process.Actuates.Buzzer.ActionTime = 1000;
            autoProc.Process.Actuates.Buzzer.DeviceID = 1;

            //Saidas
            autoProc.Process.Actuates.Saidas[0].DeviceID = 1;
            autoProc.Process.Actuates.Saidas[0].EnterDependence = 0;
            autoProc.Process.Actuates.Saidas[0].ActionTime = 1000;
```

```
autoProc.Process.Actuates.Saidas[0].ExitAction = 1;
                //...//
autoProc.Process.Actuates.Saidas[7].DeviceID = 1;
autoProc.Process.Actuates.Saidas[7].EnterDependence = 0;
autoProc.Process.Actuates.Saidas[7].ActionTime = 1000;
autoProc.Process.Actuates.Saidas[7].ExitAction = 1;

//data
autoProc.Process.DateLastExec = "10/05/2012 00:00:00";
autoProc.Process.DateInitExec = "12/06/2012 00:00:00";
autoProc.Process.DateEndExec = "13/06/2012 00:00:00";
autoProc.Process.DaysOfTheWeek = "0;0;0;0;0;0;0;0";
autoProc.Process.Periodicity = 1;
autoProc.Process.ExecHolidays = 1;
autoProc.Process.ProcessID = 0;

if (DFS.Execute(autoProc) < 0) { /*erro*/}
else
  {
     Console.WriteLine("Comando enviado com sucesso");
        }
```

Recepção de Resposta Assincrona:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
{
String BuffMsg = "";
if (args.Msg is AutoProcessResp)
{
DFS.Ack(args.Msg.DeviceID);
string  BuffMsg ="Retorno de Comandos" +
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID  " + ((AutoProcessResp)args.Msg).DeviceId.ToString()  +
"\nSequencia CMD " + ((AutoProcessResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((AutoProcessResp)args.Msg).ReturnCode.ToString()  +
"\nData Ger CMD " +    ((AutoProcessResp)args.Msg).CmdGenerationTimestamp.ToString() +
 "\nData Exe CMD  " +   ((AutoProcessResp)args.Msg).CmdExecutionTimestamp.ToString()  +
"\nTam dos Dados " + ((AutoProcessResp)args.Msg).PayloadSize.ToString();
  BuffMsg +=
  "\nTotProcess    " + ((AutoProcessResp)args.Msg).TotProcess.ToString() +
  "\n-----------------------------------------------------\n";

for(int i = 0;i < ((AutoProcessResp)args.Msg).TotProcess;i++ )
{
BuffMsg += "\nProcessCod    " +
BitConverter.ToString(((AutoProcessResp)args.Msg).Process[i].ProcessCode) +
"\nDta da ultima Execução" +
((AutoProcessResp)args.Msg).Process[i].ExeDateTime.ToString() +
"\n-----------------------------------------------------\n";
  }
Console.WriteLine(BuffMsg );
}
}
```

9.6.16. Consultar Lista de Biometrias – CheckBioList()

Descrição:

Esta função consulta a lista biométrica

Função:
Int Execute(CheckBioList);

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando
B [1]	Code	// Identificador da leitora biometrica 10,11,12

Retorno do envio de Comandos:
0 = Comando Enviado com Successo ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:

I [4]	`TotPerson`	// Quantidade de Pessoas
T	**Inicio Pessoas x [`TotPerson`]**	
S [23]	PersonID	// Identificador da pessoa Obs:Caso desejar coletar biometria utilizando o teclado : Deve conter apenas números; O identificador terá apenas 16 caracteres, preenchendo a esquerda com 7 espaços(" ") .Ex : " 0000000123456789" .
I [1]	TotCard	// Quantidade de cartões da pessoa
T	**Inicio CARTÕES x [**TotCard]	
B [1]	`TecCard`	// Tecnologia do Cartão
B [5]	CardID	// Identificador do Cartão no formato Hexadecimal
T	**Fim CARTÕES x [QtdCards]**	

Exemplo C#:

```
CheckBioList obj = new CheckBioList();
                obj.CommStatus = 2;
                obj.DeviceID = 1;
                obj.SeqCmd = 128;
                obj.Code= 4;
if (DFS.Execute(obj) < 0){ /*erro*/}
else{
                Console.WriteLine("Comando enviado com sucesso – Aguarde retorno!");
}
```

Recepção de Resposta Assincrona:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
{
String BuffMsg = "";
if (args.Msg is CheckBioResp)
{
DFS.Ack(args.Msg.DeviceID);
string  BuffMsg ="Retorno de Comandos" +
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID  " + ((CheckBioResp)args.Msg).DeviceId.ToString()  +
"\nSequencia CMD " + ((CheckBioResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((CheckBioResp)args.Msg).ReturnCode.ToString()  +
"\nData Ger CMD " + ((CheckBioResp)args.Msg).CmdGenerationTimestamp.ToString() +
 "\nData Exe CMD  " +
       ((CheckBioResp)args.Msg).CmdExecutionTimestamp.ToString()  +
```

```
"\nTam dos Dados " + ((CheckBioResp)args.Msg).PayloadSize.ToString();

BuffMsg += "\nTotal de Pessoas " + ((CheckBioResp)args.Msg).TotPerson.ToString()
+ "\n\n";
  for (int i = 0; i < ((CheckBioResp)args.Msg).TotPerson; i++)
   {
          BuffMsg += "\n Person iD " +
ascii.GetString(((CheckBioResp)args.Msg).Users[i].PersonID) +
                       "\nTotCard " +
((CheckBioResp)args.Msg).Users[i].TotCard.ToString();

          for (int j = 0; j < ((CheckBioResp)args.Msg).Users[i].TotCard; j++)
          {
                BuffMsg += "\nCardID " +
BitConverter.ToString((((CheckBioResp)args.Msg).Users[i].CardData[j].CardID)).Rep
lace("-", "").ToString() +
                                "\nTecCard " +
((CheckBioResp)args.Msg).Users[i].CardData[j].TecCard.ToString() + "\n";
          }
          BuffMsg += "-----------------------------------------------------------
---------\n";
   }
}
Console.WriteLine(BuffMsg );
}
}
```

9.6.17. Atualizar Biometria – UpdateBio()

Descrição:
Esta função atualiza a biometria.
Tipo de Operação:
1=REFAZER/INCLUIR ESTRUTURA DE PESSOA.
2=REFAZER ESTRUTURA DE CRACHÁ
3= EXCLUIR ESTRUTURA DE PESSOA

Função:
Int Execute(UpdateBio);

Parâmetros:

I[1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I[]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I[]	SeqCmd	// Sequencial do comando
I[]	Location	//Localização da leitora

B[1]	Type	//Tipo:
		1=REFAZER/INCLUIR ESTRUTURA DE PESSOA.
		2=REFAZER ESTRUTURA DE CRACHÁ
		3= EXCLUIR ESTRUTURA DE PESSOA

B[5]	CardID	//Id do crachá no formato Hexadecimal
B[2048]	Template	//Template
B [23]	PersonID	//Identificador da pessoa que possui o cartão . Obs:Caso desejar coletar biometria utilizando o teclado : Deve conter apenas números;

O identificador terá apenas 16 caracteres, preenchendo a esquerda com 7 espaços(" ") .Ex : " 0000000123456789" .

B[2] BioConfLevel //Nível de conferencia biométrico da leitora, retornar de 1-100

B [1] CardTec // Categoria:

00 - DESCONHECIDA
01 – BARRAS
02 – PROXIMIDADE
03 – SMARTCARD

Retorno do envio de Comandos:

0 = Comando Enviado com Successo ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:

Exemplo C#:

```
          UpdateBio obj = new UpdateBio();
obj.CommStatus = 2;
            obj.DeviceID = 1;
            obj.SeqCmd = 128;
            obj.Tot = 1;
            obj.Type = 1;
Operation[] operation = new  Operation[3];

//Operação 1  Type = 1
operation[1] = new Operation();
operation[1].Type = 1;
operation[1].TemplateListPerson  = new TemplateListPerson();
operation[1].TemplateListPerson.DeviceID = 1 ;
operation[1].TemplateListPerson.PersonID = "       0000000123456789".PadLeft(23);
operation[1].TemplateListPerson.Card = new TemplateListPersonCard[1];
operation[1].TemplateListPerson.Card[0].DeviceID = 1 ;
operation[1].TemplateListPerson.Card[0].CardID = new byte[] { 0x00, 0x01, 0x02,
0x03, 0x04 };
operation[1].TemplateListPerson.Card[0].CardTec = 3;
operation[1].TemplateListPerson.Template = new TemplateListPersonTemplate[1];
operation[1].TemplateListPerson.Template[0] = new TemplateListPersonTemplate();
operation[1].TemplateListPerson.Template[0].TamplateFactory = 1;
operation[1].TemplateListPerson.Template[0].Template = new byte[1024];
operation[1].TemplateListPerson.Template[0].DeviceID = 1 ;

//Operação 2 Type = 2
operation[0] = new Operation();
operation[0].Type = 2;
operation[0].EmployeeListPerson.PersonID = "       0000000123456789".PadLeft(23);
operation[0].EmployeeListPerson.BioConfLevel = 75;
operation[0].EmployeeListPerson.Card = new EmployeeListPersonCard[0];
operation[0].EmployeeListPerson.Card[0].CardID = new
       byte[]{0x00,0x01,0x02,0x03,0x04 };
operation[0].EmployeeListPerson.Card[0].CardTec = 3;
operation[0].EmployeeListPerson.Card[0].DeviceID = 1;

//Operação 3
operation[2] = new Operation();
operation[2].Type = 3;
operation[2]. PersonId.PersonID = "       0000000123456789".PadLeft(23);
operation[2].DeviceID = 1 ;
```

```
if (DFS.Execute(obj) < 0) { /*erro*/}
         else
         {
            Console.WriteLine("Comando enviado com sucesso");
        }
```

Recepção de Resposta Assincrona:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
{
String BuffMsg = "";
if (args.Msg is UpdateBioResp)
{
DFS.Ack(args.Msg.DeviceID);
string  BuffMsg ="Retorno de Comandos" +
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID  " + ((UpdateBioResp)args.Msg).DeviceId.ToString()  +
"\nSequencia CMD " + ((UpdateBioResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((UpdateBioResp)args.Msg).ReturnCode.ToString()  +
"\nData Ger CMD " +     ((UpdateBioResp)args.Msg).CmdGenerationTimestamp.ToString() +
 "\nData Exe CMD  " +   ((UpdateBioResp)args.Msg).CmdExecutionTimestamp.ToString()  +
"\nTam dos Dados " + ((UpdateBioResp)args.Msg).PayloadSize.ToString();

Console.WriteLine(BuffMsg );
}
}
```

9.6.18. Calibrar leitores manuais – HandkeyCalibrate()

Descrição:
Esta função calibra os leitores de HandKey.

Função:
Int Execute(HandKeyCalibrate);

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando

Retorno do envio de Comandos:
0 = Comando Enviado com Sucesso ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:
Não há dados de retorno

Exemplo C#:
Envio do Comando:

```
HandKeyCalibrate obj = new HandKeyCalibrate();
obj.CommStatus = 2;
```

```
            obj.DeviceID = 1;
            obj.SeqCmd = 128;
          if (DFS.Execute(obj) < 0) {
                  }
            else
          {
 Console.WriteLine(" Comando enviado com sucesso");
          }
```

Recepção de Resposta Assincrona:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
{
String BuffMsg = "";
if (args.Msg is HandkeyCalibrateResp)
{
DFS.Ack(args.Msg.DeviceID);
string  BuffMsg ="Retorno de Comandos" +
"\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
"\nDeviceID  " + ((HandkeyCalibrateResp)args.Msg).DeviceId.ToString()  +
"\nSequencia CMD " + ((HandkeyCalibrateResp)args.Msg).CmdSeq.ToString() +
"\nCodigo de ret " + ((HandkeyCalibrateResp)args.Msg).ReturnCode.ToString()  +
"\nData Ger CMD " +
      ((HandkeyCalibrateResp)args.Msg).CmdGenerationTimestamp.ToString() +
 "\nData Exe CMD  " +
      ((HandkeyCalibrateResp)args.Msg).CmdExecutionTimestamp.ToString()  +
"\nTam dos Dados " + ((HandkeyCalibrateResp)args.Msg).PayloadSize.ToString();

Console.WriteLine(BuffMsg );
}
}
```

9.6.19. Dispositivo Conectado – IsAlive()

Descrição:
Esta função faz comunicação com o device para verificar se esta ligado.

Função:
IntExecute(IsAlive);

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando

Retorno do envio de Comandos:
0 = Comando Enviado com Successo ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:
Não há dados de retorno

Exemplo C#:
Envio do Comando:

```
IsAlive obj = new IsAlive();
            obj.CommStatus = 2;
            obj.DeviceID = 1;
```

```
            obj.SeqCmd = 128;
if (DFS.Execute(autoProc) < 0) {
 /*erro*/
}
            else{
          Console.WriteLine("Comando enviado com sucesso ,Aguarde retorno");
           }
```

Recepção de Resposta Assincrona:

```
args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
      {if(((CommandReturn)args.Msg).CommandId    ==    (int)CommandReturnType.ALIVE
&& ((CommandReturn)args.Msg).ReturnCode == 1)
                  {
            Console.WriteLine("Device" + args.Msg.DeviceID.ToString() +" IS ALIVE" );
                  }
                 }
```

9.6.20. Trocar de DFS Server – UpdateDFSServer()

Descrição:
Esta função faz com que o dispositivo se conecte com o IP/Porta de outro DFSServer.
Tem por principal objetivo prover balanceamento de carga.

Função:
Int Execute(UpdateDFSServer);

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando
S [15]	Ip	// Endereço IP do novo DFSServer “172.15.9.2”
I [5]	Port	// Porta TCP do DFSServer 3232

Retorno do envio de Comandos:
0 = Comando Enviado com Sucesso ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:
Não há dados de retorno

Exemplo C#:
Envio do Comando:

```
 UpdateDFSServer obj = new UpdateDFSServer();
             obj.CommStatus = 2;
             obj.DeviceID = 1;
             obj.SeqCmd = 128;
             obj.Ip = "162.121.111.001";
             obj.Port = 3232;
           if (DFS.Execute(obj) < 0) {/*ERRO*/ }
             else  {
                Console.WriteLine(" Comando enviado com sucesso");

           }
```

9.6.21. Carga de Identificadores de Crachá – LoadBadgeIDS()

Descrição:
Esta função carrega os identificadores de uso do crachá.

Função:
Int Execute(LoadBadgeIDs);

Parâmetros:

I[1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando
B [1]	Id	// Identificador do Cracha
I [2]	Type	// Tipo do cartão

01=Empregado
02=Terceiro
03=Parceiro
04=Visitante/Grupo de Visitante
05=Outra Unidade
06=Provisório
07=Responsavel pelo Aluno
08=Crachá Mestre
09=Pacientes
10=Alunos
11=Acompanhante de Paciente
12=Autorização de Entrada
13=Candidato
99=Outros

Retorno do envio de Comandos:
0 = Comando Enviado com Sucesso ou -1 = Erro // Resposta ->MyListener sessão 6.1

Dados da Resposta do comando:
Não há dados de retorno

Exemplo C#:
Envio doComando:

```
BadgeIDReg[] bgId = new BadgeIDReg[2];
LoadBadgeIDs obj = new LoadBadgeIDs();

            obj.Reg = bgId;
            obj.Reg[0] = new BadgeIDReg();
            obj.Reg[0].Type = 1;
            obj.Reg[0].Id = 2;

            obj.Reg[1] = new BadgeIDReg();
            obj.Reg[1].Type = 3;
            obj.Reg[1].Id = 4;

            obj.SeqCmd = 1;
            obj.CommStatus = 2;
            obj.DeviceID = 1);
```

```
            if (DFS.Execute(obj) < 0) { /*erro*/}
              else{
                 Console.WriteLine("Comando Enviado com sucesso");
            }
```

Recepção de Resposta Assincrona:

```
if (args.Msg.MsgType == MessageType.MSG_SYNC_RETURN)
    {
    String BuffMsg = "";
    if (args.Msg is LoadBadgeIDsResp)
    {
    DFS.Ack(args.Msg.DeviceID);
    string  BuffMsg ="Retorno de Comandos" +
    "\nRetorno Tipo  " + (CommandReturnType)((CommandReturn)args.Msg).CommandId +
    "\nDeviceID  " + ((LoadBadgeIDsResp)args.Msg).DeviceId.ToString()  +
    "\nSequencia CMD " + ((LoadBadgeIDsResp)args.Msg).CmdSeq.ToString() +
    "\nCodigo de ret " + ((LoadBadgeIDsResp)args.Msg).ReturnCode.ToString()  +
    "\nData                Ger               CMD                "                +
       ((LoadBadgeIDsResp)args.Msg).CmdGenerationTimestamp.ToString() +
     "\nData             Exe             CMD                                 "             +
       ((LoadBadgeIDsResp)args.Msg).CmdExecutionTimestamp.ToString()  +
    "\nTam dos Dados " + ((LoadBadgeIDsResp)args.Msg).PayloadSize.ToString();

    Console.WriteLine(BuffMsg );
    }
    }
```

9.7. Comandos para Gerenciamento de Dispositivos de Ponto

Entende-se sob comando as ações executadas pela APP que através do DFS são validadas e enviadas ao FWR, cujas respostas podem ser Síncronas e Assíncronas.
Nesta sessão serão listados os comandos comuns aos dipositivos de Ponto: DIGIREP e DIGICP.
Em C# os erros serão reprensentados pelo enumerator "ClockResultType", em Java "ClockResultTypeJava".

Tabela de erros.

Erro	Descrição
UNKNOWN	Erro desconhecido
SUCESSFULL	Operação realizada com sucesso
MT_ERROR	Erro ao acessar o banco de dados da MT
COMPANY_IS_ALREADY_REGISTERED	Empresa já cadastrada
PERSON_IS_ALREADY_REGISTERED	Colaborador já cadastrado
NOT_FOUND_PERSON	Cadastro do colaborador não encontrado
ERROR_REMOVE_BIOMETRIC	Não foi possível remover biometria
NOT_FOUND_BIOMETRIC_READER	Leitor biométrico não encontrado
ERROR_WRITE_MRP	Erro durante a gravação do registro na MRP
COULD_NOT_UPDATE_CLOCK	Não foi possível ajustar data e hora
NEED_INSERT_COMPANY	Empresa não cadastrada
READER_IS_NOT_AVAILABLE	Leitor não disponível
COULD_NOT_UPDATE_REGISTER	Não foi possível atualizar o cadastro
READERS_ARE_ALREADY_RECORD	Cartão já cadastrado

MT_IS_FULL	Banco de dados da MT está cheio
INVALID_PIS	PIS informado é inválido
COMPANY_NAME_SIZE_IS_INVALID	Tamanho do nome da empresa inválido (0 = x > 150)
WORK_PLACE_NAME_SIZE_IS_INVALID	Tamanho do nome do local de prestação de serviço inválido (0 = x > 100)
INVALID_CLOCK	Formato da data e hora inválido
ERROR_TO_SAVE_PERSON_DATA	Não foi possível cadastrar os dados do colaborador
NOT_FOUND_RFID_READER	Leitor de proximidade não encontrado
NOT_FOUND_BARCODE_READER	Leitor de código de barras não encontrado
NOT_FOUND_MIFARE_READER	Leitor Mifare não encontrado
BIOMETRIC_IS_NOT_REGISTERED	Biometria não cadastrada
PIS_IS_IN_USE	PIS já esta em uso por outro colaborador
NOT_FOUND_BIOMETRIC_MODULE	Módulo de leitor biométrico não esta presente
COULD_NOT_REMOVE_BIOMETRIC_TEMPLATE	Não encontrada biometria para o colaborador informado
ERROR_TO_ACCESS_INER_DATABASE_OF_BIOMETRIC_READER	Erro de acesso ao banco de dados interno do leitor
ERROR_ON_PROTOCOL	Quando houver algum erro no protocolo
UNAVAIlABLE_1_TO_1	Não é possível configurar o DIGIREP para atuar em modo 1:1(ocorre caso o DIGIREP não possua um leitor biométrico)

9.7.1. Empresa – ManagerCompany()

Descrição:
Esta função insere uma empresa no relógio, caso já exista e o CNPJ ou Razão Social ou CEI for diferente, esta é atualizada.
A cada movimentação de inclusão ou alteração nas condições do parágrafo anterior será incluído um registro do tipo 2 na MRP.

Função:
Int Execute(ManagerCompany);

Parâmetros:

I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
S[]	Cei	// CEI da empresa.
S[]	Identier	// Identificador, CPF ou CNPJ.
S[]	SocialName	// Nome da razão social.
S[]	Local	// Nome do local onde esta afixado o equipamento
E	Status	// Valores referentes a Enumerator ClockResultType ou ClockResultTypeJava com valores de resultados, veja valores na **Tabela de Erros.**

Exemplo C#:
Envio do Comando:

```
ManagerCompany obj = new ManagerCompany();
       obj.DeviceID = 1;
       obj.Company.Cei = "000000000000";
       obj.Company.Identifier = "000000000";
       obj.Company.SocialName = "Empresa";
       obj.Company.Local = "São Paulo";

       DFS.Execute(obj);
```

Recepção de Resposta Assincrona

```
if (args.Msg is InsertUpdateCompanyResp)
       {
       String BuffMsg=
                "\nRetorno do comando InsertUpdateCompany " +
                "\nDeviceID " +
       ((InsertUpdateCompanyResp)args.Msg).DeviceID.ToString() +
                "\nStatus " +
       (ClockResultType)((InsertUpdateCompanyResp)args.Msg).Status;

       DFS.Ack(args.Msg.DeviceID);

       Console.WriteLine(msg );

       }
```

9.7.2. Solicita os Eventos Não Enviados – RequestLog()

Descrição:
Esta função solicita os últimos registros não enviados da área temporária e limpa o buffer após confirmação de aceite pela APP.

Os FWR dos dispositivos de ponto, além de gravar o evento em área protegida MRP, escrevem simultaneamente em duas áreas: temporária (eventos ainda não enviados) e backup (circular acima explicado).

Função:
Int Execute(RequestLog);

Parâmetros:

I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I[ ]	TotEvent	//Total de eventos.
D	TimeEvent	//Data em que ocorreu o evento.
I	Nsr	// NSR da marcação.
L[]	Pis	//Pis
S[30]	PersonId	//Identificador da pessoa.
E	EventId	//Id's dos eventos, vide item **EventosClock.**

Exemplo C#:
Envio doComando:

```
           RequestLog obj = new RequestLog();
           obj.DeviceID = devId;

           DFS.Execute(obj);
```

Recepção de Resposta Assincrona

```
else if (args.Msg is RequestLogResp)
            {

                msg = "\nEvento....= RequestLogResp " +
                    "\nDeviceID.......= " +
((RequestLogResp)args.Msg).DeviceID.ToString() +
                   "\nTot Event.....= " +
((RequestLogResp)args.Msg).TotEvent.ToString() + "\n";

                DFS.Ack(args.Msg.DeviceID);

                for (int i = 0, n = 1; i <
(int)((RequestLogResp)args.Msg).TotEvent; i++, n++)
                {
                    msg += "\nEvento nº " + n.ToString() +
                        "\nId Evento  = " +
(EventClockType)((RequestLogResp)args.Msg).Events[i].EventID +
                              "\nDate    = " +
((RequestLogResp)args.Msg).Events[i].TimeEvent.ToString() + "\n";

     if ((EventClockType)((RequestLogResp)args.Msg).Events[i].EventID ==
EVENTS.REGISTERED_POINT)
                    {
                        msg += "\nNSR " +
((EventRegisteredPoint)((RequestLogResp)args.Msg).Events[i]).Nsr.ToString() +
                          "\nPis " +
((EventRegisteredPoint)((RequestLogResp)args.Msg).Events[i]).Pis.ToString() +
                          "\nTec reader " +
(CARDTECHTYPE)((EventRegisteredPoint)((RequestLogResp)args.Msg).Events[i]).Techno
logy + "\n";
                    }

                    msg += "\n";
                }

                Console.WriteLine(msg );
            }
```

9.7.3. Solicita Eventos da Área de Backup – RequestBackupLog()

Descrição:
Esta função solicita os últimos registros 256 K (15.000 considerando apenas ponto) de log de eventos da área de backup que é circular.
Os FWR dos dispositivos de ponto, além de gravar o evento em área protegida MRP, escrevem simultaneamente em duas áreas: temporária (eventos ainda não enviados) e backup (circular acima explicado).

Função:
Int Execute(RequestBackupLog);

Parâmetros:

I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I[ ]	TotEvent	//Total de eventos.
D	TimeEvent	//Data em que ocorreu o evento.

I	Nsr	// NSR da marcação.
L[]	Pis	//Pis
S[30]	PersonId	//Identificador da pessoa.
E	EventId	//Id's dos eventos, vide item **EventosClock.**

Exemplo C#:
Envio doComando:

```
RequestBackupLog obj = new RequestBackupLog();
       obj.DeviceID = 1;

       DFS.Execute(obj);
```

Recepção de Resposta Assincrona

```
if (args.Msg is RequestBackupLogResp)
       {
           msg = "\nEvento....= RequestBackupLogResp " +
              "\nDeviceID.......=                    "                    +
       ((RequestBackupLogResp)args.Msg).DeviceID.ToString() +
              "\nTot                Event.....=                 "          +
       ((RequestBackupLogResp)args.Msg).TotEvent.ToString() + "\n";

              DFS.Ack(args.Msg.DeviceID);

              for      (int      i      =      0,      n      =      1;      i      <
(int)((RequestBackupLogResp)args.Msg).TotEvent; i++, n++)
              {
                     msg += "\nEvento nº " + n.ToString() + "\nId Evento   = " +
       (EventClockType)((RequestBackupLogResp)args.Msg).Events[i].EventID  +
                     "\nDate                                    =        "      +
       ((RequestBackupLogResp)args.Msg).Events[i].TimeEvent.ToString() +   "\n";

                     if
((EventClockType)((RequestBackupLogResp)args.Msg).Events[i].EventID            ==
EVENTS.REGISTERED_POINT)
                     {
                     msg            +=            "\nNSR            "            +
((EventRegisteredPoint)((RequestBackupLogResp)args.Msg).Events[i]).Nsr.ToString()
+
                       "\nPis                          "                        +
((EventRegisteredPoint)((RequestBackupLogResp)args.Msg).Events[i]).Pis.ToString()
+
                       "\nTec              reader               "              +
(CARDTECHTYPE)((EventRegisteredPoint)((RequestBackupLogResp)args.Msg).Events[i]).
Technology + "\n";
                     }

                     msg += "\n";
              }
           Console.WriteLine(msg );
       }
```

9.7.4. Solicita Informações do Colaborador – RequestPersonList()

Descrição:
Esta função requisita a lista total do dispositivo ou uma pessoa específica da lista.

Função:
Int Execute(RequestPersonList);

Parâmetros:

I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I[ ]	RequisitionType	// Requisita uma pessoa = 1 ou requisita a lista completa = 0.
S[30]	PersonID	// Id da pessoa a ser consultada.

Exemplo C#:
Envio do Comando:

```
RequestPersonList obj = new RequestPersonList();
       obj.DeviceID = 1;
       obj.PersonID = "Jesus".PadRight(30, ' ');
       obj.RequisitionType = 1;

       DFS.Execute(obj);
```

Recepção de Resposta Assincrona

```
if (args.Msg is RequestPersonListResp)
       {
       DFS.Ack(args.Msg.DeviceID);

       msg += "\nRetorno do comando RequestPeopleList " +
              "\n\nDeviceID " +
       ((RequestPersonListResp)args.Msg).DeviceID.ToString() +
       "\nTotPerson " +
        ((RequestPersonListResp)args.Msg).Totperson.ToString() + "\n\n\n";

       for (int i = 0; i < ((RequestPersonListResp)args.Msg).Totperson; i++)
              {
              msg += i.ToString() +
              "\nName " + ((RequestPersonListResp)args.Msg).Person[i].Name +
              "\nPersonID " +
       ((RequestPersonListResp)args.Msg).Person[i].PersonID +
              "\nPis " +
       ((RequestPersonListResp)args.Msg).Person[i].Pis.ToString() +
              "\nTypeOfValidation " +
       ((RequestPersonListResp)args.Msg).Person[i].TypeOfValidation.ToString() +
"\n";

       for (int x = 0; x <
       ((RequestPersonListResp)args.Msg).Person[i].CardClock.Length; x++)
              {
              msg += "\nCardId " +
       ((RequestPersonListResp)args.Msg).Person[i].CardClock[x].CardID.ToString()
+
              "\nCardTec " +
       (CardTechType)((RequestPersonListResp)args.Msg).Person[i].CardClock[x].Car
dTec + "\n"; CardTechType

              }
              msg += "-----------------------------------------------\n\n";
              }

       Console.WriteLine(msg );
       }
```

9.8. Requisições e Respostas dos Dispositivos de Acesso

Nesta sessão será descrito como a aplicação deverá lidar com os eventos de **acesso**.

Segue abaixo as etapas necessárias para atender eventos de acesso com o Firmware Digicon no estado de ON_LINE:

1) Quando um usuário passa seu cartão sobre uma leitora, ou digita sua senha em um teclado, coloca seu dedo em um dispositivo leitor biométrico de dedo, ou efetua quaisquer operações que exijam liberação de acesso, estando o periférico ON_LINE o mesmo irá enviar ao DFS um evento de requisição de acesso com essas informações.
2) O DFS ao receber este evento o repassa imediatamente à aplicação através de seu call-back/listener.
3) A Aplicação deve imediatamente processar esta requisição e com essas informações decidir se deve autorizar ou negar o acesso.
4) Após a decisão (Autorizar ou Negar) a aplicação deve responder rapidamente à requisição de acesso.
5) Essa resposta à requisição de acesso passa pelo DFS que formata e encaminha para o Firmware.
6) O Firmware ao receber a resposta toma a ação determinada pela aplicação e em seguida gera um evento reportando à aplicação que fez o que foi determinado pela aplicação.
7) A Aplicação então recebe este evento e registra em seu banco de dados.

São 27 (vinte e sete) respostas possíveis para requisições de acesso recebidas pelo DFS através de crachá, Tipo =1 (`MSG_ACCESS_REQUEST_CARD`) e pessoa (matrícula), Tipo=2 (`MSG_ACCESS_REQUEST_PERSON`);

A seguir serão apresentadas as respostas baseada no Tipo = 2 (pessoa),quando o Tipo = 1(crachá), sua resposta é a mais completa, que incluirá o bloco de **Pendências**.

9.8.1. Acesso Válido

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
B[5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
I[1]	EventID	//1=Acesso Válido Crachá
B[5]	PersonLogicalId	// Identificador lógico do cartão da pessoa (Número da matrícula)
B[1]	CardTech	// Categoria: 00 - DESCONHECIDA 01 – BARRAS 02 – PROXIMIDADE 03 – SMARTCARD

B [1]	CardType	//Tipos: 01 – EMPREGADO 02 – TERCEIRO 03 – PARCEIRO 04 - VISITANTE, GRUPO DE VISITANTES 05 – OUTRA UNIDADE 06 – PROVISÓRIO 07 – RESPONSÁVEL PELO ALUNO 08 – CRACHÁ MESTRE 09 – PACIENTES 10 – ALUNOS 11 – ACOMPANHANTE DE PACIENTES 12 – AUTORIZAÇÃO DE ENTRADA 13 – CANDIDATO 99 - OUTROS
B [1]	ActualLevel	//Nível atual da pessoa 1 a 98, se responder 99 indicará ao firmware que pode passar em qualquer sentido observando os valores do Level1 e Level2 da leitora
S[16]	TimeScheduleCredit	//Faixa Hora decremento credito “hh:mm–hh:mm”
B[1]	BiometricLevel	//Nivel de Conferencia Biometrica 0(no) a 9(Hi)
S[6]	Password	//Senha do usuário 06 digitos 000000 a 999999
S[3]	DataUpdateFlag	//Flag ex: “0;0” 1º: Se é necessário atualizar a data de controle de intervalo de almoço e/ou interjornada. 2º: se é necessário atualizar a data de última passagem no refeitório.
B [1]	CreditControlType	//Tipo de Controle de Credito 0=nenhum.
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID
B [1]	Frisk	Revista (0 – não revistar; 1 – revista obrigatória; 2 – sortear)
I [9]	DeviceID	//ID do dispositivo para onde deverá ser enviado o comando
B [23]	PersonId	//Identificador da pessoa que possui o cartão . Obs:Caso desejar coletar biometria utilizando o teclado : Deve conter apenas números; O identificador terá apenas 16 caracteres, preenchendo a esquerda com 7 espaços(" ") .Ex : " 0000000123456789" .
B [1]	FieldID	//Identificador do campo
B [1]	Data	//Dados
I [1]	ResponseSize	// Quantidade de bytes dos dados, **Não setar esse campo.**
I [1]	PendingStructSize	//Tamanho do bloco de pêndencias
D	LastUpdate	// Data da ultima atualização do crachá
D	TimestampRequest	//Data do acesso

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {
```

```
//Para pegar o id da pessoa – matricula ou identificador
Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;
int Deviceid = ((AccessRequestPerson)args.Msg).DeviceID;
int Readerid = ((AccessRequestPerson)args.Msg).ReaderId;

// Cabeçalho da Resposta
ResponseAccessValid rav = new ResponseAccessValid();
       rav.DeviceID = args.Msg.DeviceID;
              rav.ActualLevel = 0;
rav.BiometricLevel = 0;
rav.CreditControlType = 1;
rav.TimeScheduleCredit = 1;
rav.Frisk = 0;
rav.Password = "123456";
rav.DataUpdateFlag = 0;
              rav.UserMessage=
                     Encoding.ASCII.GetBytes("hello".PadRight(32));

//PayLoad da Resposta
ResponsePersonStructure rts = new ResponsePersonStructure();
rts.EventStructure = rav;
rts.DeviceID = args.Msg.DeviceID;
rts.AppConnectionStatus = 2;
rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
rts.UserMessage = new Byte[32];

DFS.Execute(rts);

}
```

Abaixo a resposta para o tipo de requisição de crachá acesso válido:

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_CARD )
       {

       Byte[] bt = (Byte[])((AccessRequestCard)args.Msg).CardId;
       int Deviceid = ((AccessRequestCard)args.Msg).DeviceID ;
       int Readerid = ((AccessRequestCard)args.Msg).ReaderId ;

// Cabeçalho da Resposta
       ResponseAccessValid rav = new ResponseAccessValid();
              rav.DeviceID = args.Msg.DeviceID;
                     rav.ActualLevel = 0;
       rav.BiometricLevel = 0;
       rav.CreditControlType = 1;
       rav.TimeScheduleCredit = 1;
       rav.Frisk = 0;
       rav.Password = "123456";
       rav.DataUpdateFlag = 0;
       rav.UserMessage=Encoding.ASCII.GetBytes("hello".PadLeft(32));
//Fim do Cabeçalho da Resposta

//Início do bloco de pendências
      //Pendências de :
      // Faixa horária
      PendingStructure pds = new PendingStructure();
      pds.PendingData = new PendingData[1];
      pds.PendingData[0] = new PendingData();
```

```
      pds.PendingData[0].DeviceID = args.Msg.DeviceID;
      pds.PendingData[0].FieldID = 35; //campo

     int inicio = 800; //
     int fim = 1200; //max = 1439 minutos

     byte[] ranges = new byte[21];
     ranges[0] = (byte)(inicio & 0X00FF);
     ranges[1] = (byte)(((inicio & 0X0F00) >> fim) | (fim & 0X0F00) >> 8);
     ranges[2] = (byte)(fim & 0X0FF);

     pds.PendingData[0].Data = ranges;
     pds.DeviceID = args.Msg.DeviceID;
     pds.LastUpdate = DateTime.Now;
```
//Fim do bloco de pendências

// PayLoad da Resposta de Cartão

```
      ResponseTagStructure rts = new ResponseTagStructure();
      rts.PendingStructure = pds; //bloco de pendências
       rts.PendingStructSize = pds.Bytes.Length; //tamanho do pendências
       rts.PersonId = Encoding.ASCII.GetBytes("0000000123456789".PadLeft(23));
       rts.PersonlogicalId = ((AccessRequestCard)args.Msg).CardId;
       rts.CardTech = 3;
       rts.CardType = 8;
       rts.EventStructure = rav;
       rts.AppConnectionStatus = 2;
       rts.UserMessage = new Byte[32];
       rts.DeviceID = args.Msg.DeviceID;
```
//Fim do PayLoad da Resposta de Cartão

```
       DFS.Execute(rts);
       }
```

Note que na resposta para o tipo crachá, o que muda é o payload da resposta, o cabeçalho é o mesmo, isso é válido para todas as respostas nesta seção 8.9.

9.8.2. Acesso Permitido Autorizador

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I [1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
I[1]	EventID	// 45=Acesso Permitido Autorizador
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
B [1]	CardTech	// Categoria: 00 - DESCONHECIDA 01 – BARRAS 02 – PROXIMIDADE 03 – SMARTCARD

B [1] CardType		//Tipos: 01 – EMPREGADO 02 – TERCEIRO 03 – PARCEIRO 04 - VISITANTE, GRUPO DE VISITANTES 05 – OUTRA UNIDADE 06 – PROVISÓRIO 07 – RESPONSÁVEL PELO ALUNO 08 – CRACHÁ MESTRE 09 – PACIENTES 10 – ALUNOS 11 – ACOMPANHANTE DE PACIENTES 12 – AUTORIZAÇÃO DE ENTRADA 13 – CANDIDATO 99 - OUTROS
B [1]	ActualLevel	//Nível atual da pessoa 1 a 99
S [16]	TimeScheduleCredit	//Faixa Hora decremento credito “hh:mm–hh:mm”
B [1]	BiometricLevel	//Nivel de Conferencia Biometrica 0(no) a 9(Hi)
S [6]	Password	//Senha do usuário 06 digitos 000000 a 999999
S [3]	DataUpdateFlag	//Flag ex: “0;0” 1º: Se é necessário atualizar a data de controle de intervalo de almoço e/ou interjornada. 2º: se é necessário atualizar a data de última passagem no refeitório.
B [1]	CreditControlType	//Tipo de Controle de Credito 0=nenhum.
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID
B [1]	Frisk	Revista (0 – não revistar; 1 – revista obrigatória; 2 – sortear)
I [9]	DeviceID	//ID do dispositivo para onde deverá ser enviado o comando
B [23]	PersonId	//Identificador da pessoa que possui o cartão . Obs:Caso desejar coletar biometria utilizando o teclado : Deve conter apenas números; O identificador terá apenas 16 caracteres, preenchendo a esquerda com 7 espaços(" ") .Ex : " 0000000123456789"
.		
B [1]	FieldID	//Identificador do campo
B [1]	Data	//Dados
I [1]	ResponseSize	// Quantidade de bytes dos dados, **Não atribuir valor a esse campo.**
I [1]	PendingStructSize	//Tamanho do bloco de pêndencias
D	LastUpdate	// Data da ultima atualização do crachá
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;
        int Deviceid = ((AccessRequestPerson)args.Msg).DeviceID ;
        int Readerid = ((AccessRequestPerson)args.Msg).ReaderId ;

        // Cabeçalho da Resposta
        ResponseAccessValidAuthorizer rav = new ResponseAccessValidAuthorizer();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;
        rav.BiometricLevel = 0;
        rav.CreditControlType = 1;
        rav.TimeScheduleCredit = 1;
        rav.Frisk = 0;
        rav.Password = "123456";
        rav.DataUpdateFlag = 0;
                    rav.UserMessage=
                           Encoding.ASCII.GetBytes("hello".PadRight(32));

        //PayLoad  da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                    rts.EventStructure = rav;
                    rts.DeviceID = args.Msg.DeviceID;
                    rts.AppConnectionStatus = 2;
        rts.UserMessage = new Byte[32];
rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };

        DFS.Execute(rts);

        }
```

9.8.3. Liberação Incondicional – Cartão Mestre

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I [1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
I[1]	EventID	// 11=Liberado por cartão mestre
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
B [1]	CardTech	// Categoria: 00 - DESCONHECIDA 01 – BARRAS 02 – PROXIMIDADE 03 – SMARTCARD
B [1]	CardType	//Tipos: 01 – EMPREGADO

		02 – TERCEIRO 03 – PARCEIRO 04 - VISITANTE, GRUPO DE VISITANTES 05 – OUTRA UNIDADE 06 – PROVISÓRIO 07 – RESPONSÁVEL PELO ALUNO 08 – CRACHÁ MESTRE 09 – PACIENTES 10 – ALUNOS 11 – ACOMPANHANTE DE PACIENTES 12 – AUTORIZAÇÃO DE ENTRADA 13 – CANDIDATO 99 - OUTROS
B [1]	ActualLevel	//Nível atual da pessoa 1 a 99
S [16]	TimeScheduleCredit	//Faixa Hora decremento credito “hh:mm–hh:mm”
B [1]	BiometricLevel	//Nivel de Conferencia Biometrica 0(no) a 9(Hi)
S [6]	Password	//Senha do usuário 06 digitos 000000 a 999999
S [3]	DataUpdateFlag	//Flag ex: “0;0” 1º: Se é necessário atualizar a data de controle de intervalo de almoço e/ou interjornada. 2º: se é necessário atualizar a data de última passagem no refeitório.
B [1]	CreditControlType	//Tipo de Controle de Credito 0=nenhum.
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID
B [1]	Frisk	Revista (0 – não revistar; 1 – revista obrigatória; 2 – sortear)
I [9]	DeviceID	//ID do dispositivo para onde deverá ser enviado o comando
B [23]	PersonId	//Identificador da pessoa que possui o cartão . Obs:Caso desejar coletar biometria utilizando o teclado : Deve conter apenas números; O identificador terá apenas 16 caracteres, preenchendo a esquerda com 7 espaços(" ") .Ex : " 0000000123456789" .
B [1]	FieldID	//Identificador do campo
B [1]	Data	//Dados
I [1]	ResponseSize	// Quantidade de bytes dos dados, **Não atribuir valor a esse campo.**
I [1]	PendingStructSize	//Tamanho do bloco de pêndencias
D	LastUpdate	// Data da ultima atualização do crachá
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
```

```
Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

// Cabeçalho da Resposta
ResponseAccessValidMasterCard rav = new ResponseAccessValidMasterCard();
            rav.DeviceID = args.Msg.DeviceID;
            rav.ActualLevel = 0;
            rav.TimeScheduleCredit = 1;
            rav.BiometricLevel = 0;
            rav.Frisk = 0;
            rav.Password = "123456";
            rav.DataUpdateFlag = 0;
            rav.CreditControlType = 0;
            rav.UserMessage = new Byte[32];

//PayLoad  da Resposta
ResponsePersonStructure rts = new ResponsePersonStructure();
            rts.EventStructure = rav;
            rts.DeviceID = args.Msg.DeviceID;
            rts.AppConnectionStatus = 2;
            rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
            rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
            rts.UserMessage = new Byte[32];

DFS.Execute(rts);

}
```

9.8.4. Acesso Negado Aguardando Autorizador

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I [1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
I[1]	EventID	// 47=Acesso Negado Aguardando Autorizador
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
B [1]	CardTech	// Categoria: 00 - DESCONHECIDA 01 – BARRAS 02 – PROXIMIDADE 03 – SMARTCARD
B [1]	CardType	//Tipos: 01 – EMPREGADO 02 – TERCEIRO 03 – PARCEIRO 04 - VISITANTE, GRUPO DE VISITANTES 05 – OUTRA UNIDADE 06 – PROVISÓRIO 07 – RESPONSÁVEL PELO ALUNO 08 – CRACHÁ MESTRE 09 – PACIENTES 10 – ALUNOS

11 – ACOMPANHANTE DE PACIENTES
12 – AUTORIZAÇÃO DE ENTRADA
13 – CANDIDATO
99 - OUTROS

B [1]	ActualLevel	//Nível atual da pessoa 1 a 99
S [16]	TimeScheduleCredit	//Faixa Hora decremento credito “hh:mm–hh:mm”
B [1]	BiometricLevel	//Nivel de Conferencia Biometrica 0(no) a 9(Hi)
S [6]	Password	//Senha do usuário 06 digitos 000000 a 999999
S [3]	DataUpdateFlag	//Flag ex: “0;0” 1º: Se é necessário atualizar a data de controle de intervalo de almoço e/ou interjornada. 2º: se é necessário atualizar a data de última passagem no refeitório.
B [1]	CreditControlType	//Tipo de Controle de Credito 0=nenhum.
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID
B [1]	Frisk	//Revista (0 – não revistar; 1 – revista obrigatória; 2 – sortear)
I [9]	DeviceID	//ID do dispositivo para onde deverá ser enviado o comando
B [23]	PersonId	//Identificador da pessoa que possui o cartão . Obs:Caso desejar coletar biometria utilizando o teclado : Deve conter apenas números; O identificador terá apenas 16 caracteres, preenchendo a esquerda com 7 espaços(" ") .Ex : " 0000000123456789" .
B [1]	FieldID	//Identificador do campo
B [1]	Data	//Dados
I [1]	ResponseSize	// Quantidade de bytes dos dados. **Não atribuir valor a esse campo.**
I [1]	PendingStructSize	//Tamanho do bloco de pêndencias
D	LastUpdate	// Data da ultima atualização do crachá
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da Resposta
        ResponseAccessDeniedAwatingAuthorizer rav = new
               ResponseAccessDeniedAwatingAuthorizer();
                      rav.DeviceID = args.Msg.DeviceID;
                      rav.ActualLevel = 0;
                      rav.TimeScheduleCredit = 1;
                      rav.BiometricLevel = 0;
                      rav.Frisk = 0;
                      rav.Password = "123456";
```

```
                rav.DataUpdateFlag = 0;
                rav.CreditControlType = 0;
                rav.UserMessage = new Byte[32];

        //PayLoad  da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                rts.EventStructure = rav;
                rts.DeviceID = args.Msg.DeviceID;
                rts.AppConnectionStatus = 2;
                rts.UserMessage = new Byte[32];
                rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };

        DFS.Execute(rts);

        }
```

9.8.5. Acesso Negado pela Faixa Horária da Permissão

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 13=Acesso Negado faixa horária permissão
B [1]	ActualLevel	//Nível atual da pessoa 1 a 99
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Estrutura de Eventos:

B[1]	PersonLevel	//Nível atual da pessoa 1 a 99

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedPermissionHourlyRange rav = new
                ResponseAccessDeniedPermissionHourlyRange();
                rav.DeviceID = args.Msg.DeviceID;
                rav.ActualLevel = 0;
```

```
// PayLoad da Resposta
ResponsePersonStructure rts = new ResponsePersonStructure();
rts.EventStructure = rav;
rts.DeviceID = args.Msg.DeviceID;
rts.AppConnectionStatus = 2;
rts.UserMessage = new Byte[32];
rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };

DFS.Execute(rts);

}
```

9.8.6. Acesso Negado pelo Tipo de Pessoa

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 9=Acesso Negado Tipo de pessoa
B [1]	ActualLevel	//Nível atual da pessoa 1 a 99
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;


        // Cabeçalho da resposta
        ResponseAccessDeniedPersonType rav = new ResponseAccessDeniedPersonType();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;

        // PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
        rts.EventStructure = rav;
        rts.DeviceID = args.Msg.DeviceID;
        rts.AppConnectionStatus = 2;
```

```
rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
rts.UserMessage = new Byte[32];


DFS.Execute(rts);
}
```

9.8.7. Acesso Negado Permissão

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 2=Acesso Negado permissão
B[1]	ActualLevel	//Nível atual da pessoa 1 a 99
I[9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B[5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B[5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedPermission rav = new ResponseAccessDeniedPermission();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;

        // PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
        rts.EventStructure = rav;
        rts.DeviceID = args.Msg.DeviceID;
        rts.AppConnectionStatus = 2;
        rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
        rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
        rts.UserMessage = new Byte[32];
```

```
DFS.Execute(rts);

}
```

9.8.8. Acesso Negado Situação

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 3=Acesso Negado Situação
B[1]	ActualLevel	//Nível atual da pessoa 1 a 99
I[9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B[5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B[5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedSituation rav = new ResponseAccessDeniedSituation();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;

        // PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
        rts.EventStructure = rav;
        rts.DeviceID = args.Msg.DeviceID;
        rts.AppConnectionStatus = 2;
        rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
        rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
        rts.UserMessage = new Byte[32];


        DFS.Execute(rts);

        }
```

9.8.9. Acesso Negado Crédito de Acesso

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 7=Acesso Negado Credito de acesso
B [1]	ActualLevel	//Nível atual da pessoa 1 a 99
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
	{

	//Para pegar o id da pessoa – matricula ou identificador
	Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

	// Cabeçalho da resposta
	ResponseAccessDeniedCredit rav = new ResponseAccessDeniedCredit();
			rav.DeviceID = args.Msg.DeviceID;
			rav.ActualLevel = 0;


// PayLoad da Resposta
	ResponsePersonStructure rts = new ResponsePersonStructure();
			rts.EventStructure = rav;
			rts.DeviceID = args.Msg.DeviceID;
			rts.AppConnectionStatus = 2;
			rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
			rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
			rts.UserMessage = new Byte[32];


	DFS.Execute(rts);

	}
```

9.8.10. Acesso Negado pela Faixa Horária da Pessoa

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 8=Acesso Negado faixa horária pessoa
B [1]	ActualLevel	//Nível atual da pessoa 1 a 99
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedPersonHourlyRange rav = new
               ResponseAccessDeniedPersonHourlyRange();
                      rav.DeviceID = args.Msg.DeviceID;
                      rav.ActualLevel = 0;i

        // PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                      rts.EventStructure = rav;
                      rts.DeviceID = args.Msg.DeviceID;
                      rts.AppConnectionStatus = 2;
                      rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                      rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                      rts.UserMessage = new Byte[32];


        DFS.Execute(rts);

        }
```

9.8.11. Acesso Negado na Saída pelo Tipo de Pessoa

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 22=Acesso Negado Saida tipo pessoa
B [1]	ActualLevel	//Nível atual da pessoa 1 a 99
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedPersonTypeExit rav = new
               ResponseAccessDeniedPersonTypeExit();
                      rav.DeviceID = args.Msg.DeviceID;
                      rav.ActualLevel = 0;

        // PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                      rts.EventStructure = rav;
                      rts.DeviceID = args.Msg.DeviceID;
                      rts.AppConnectionStatus = 2;
                      rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                      rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                      rts.UserMessage = new Byte[32];


        DFS.Execute(rts);

        }
```

9.8.12. Acesso Negado Intervalo de Almoço

Função:

int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 24=Acesso Negado Intervaço almoço

B [1]	ActualLevel	//Nível atual da pessoa 1 a 99
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedLunchInterval rav = new
        ResponseAccessDeniedLunchInterval();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;

        // PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                    rts.EventStructure = rav;
                    rts.DeviceID = args.Msg.DeviceID;
                    rts.AppConnectionStatus = 2;
                    rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                    rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                    rts.UserMessage = new Byte[32];


        DFS.Execute(rts);

        }
```

9.8.13. Acesso Negado pelo Bloqueio por Falta

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 25=Acesso Negado por falta
B [1]	ActualLevel	//Nível atual da pessoa 1 a 99

I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON){

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedFault rav = new ResponseAccessDeniedFault();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;



        //PayLoad  da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                    rts.EventStructure = rav;
                    rts.DeviceID = args.Msg.DeviceID;
                    rts.AppConnectionStatus = 2;
                    rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};

                    rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                    rts.UserMessage = new Byte[32];


        DFS.Execute(rts);

        }
```

9.8.14. Acesso Negado pela Interjornada

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 32=Acesso Negado pela interjornada
B [1]	ActualLevel	//Nível atual da pessoa 1 a 99
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando

B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedInterJourney rav = new
        ResponseAccessDeniedInterJourney();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;

        // PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                    rts.EventStructure = rav;
                    rts.DeviceID = args.Msg.DeviceID;
                    rts.AppConnectionStatus = 2;
                    rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                    rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                    rts.UserMessage = new Byte[32];


        DFS.Execute(rts);

        }
```

9.8.15. Acesso Negado fora da Faixa de Crédito de Acesso

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 49=Acesso Negado Fora faixa credito
B [1]	ActualLevel	//Nível atual da pessoa 1 a 99
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa

		(Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedCreditRange rav = new
        ResponseAccessDeniedCreditRange();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;

        //PayLoad  da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                    rts.EventStructure = rav;
                    rts.DeviceID = args.Msg.DeviceID;
                    rts.AppConnectionStatus = 2;
                    rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                    rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                    rts.UserMessage = new Byte[32];

        DFS.Execute(rts);
        }
```

9.8.16. Acesso Negado Validade do Crachá

Função:

int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 13=Acesso Negado faixa horária permissão
B[4]	LastBadgeValidate	// Data Validade do Cracha dd/mm/aaaa
B [1]	ActualLevel	//Nível atual da pessoa 1 a 99
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B[5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B[5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso

B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedValidity rav = new ResponseAccessDeniedValidity();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;
                    rav.TagLastValidDate = Convert.ToDateTime("31/05/2012
    00:00:00");

        // PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                    rts.EventStructure = rav;
                    rts.DeviceID = args.Msg.DeviceID;
                    rts.AppConnectionStatus = 2;
                    rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                    rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                    rts.UserMessage = new Byte[32];


        DFS.Execute(rts);

        }
```

9.8.17. Acesso Negado Nivel de Controle

Função:

int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 5=Acesso Negado Nivel de Controle

I[1]	DenyType	// Tipo da Negação:
		26 = Acesso Negado na Tentativa de Entrar sem ter Saído
		28 = Acesso Negado na Tentativa de Sair sem ter Entrado
		48 = Tentativa de Sair do Nível1 sem ter Saído do Nível2
		49 = Tentativa de Entrar em Nível2 sem ter Entrado Nível1

B [1]	ActualLevel	//Nível atual da pessoa 1 a 99

I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedLevelControl rav = new
        ResponseAccessDeniedLevelControl();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;

        // PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                    rts.EventStructure = rav;
                    rts.DeviceID = args.Msg.DeviceID;
                    rts.AppConnectionStatus = 2;
                    rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                    rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                    rts.UserMessage = new Byte[32];

        DFS.Execute(rts);
        }
```

9.8.18. Acesso Negado por AntiDupla

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 48=AntiDupla

I[1]	DenyType	// Tipo da Negação:
		26 = Acesso Negado na Tentativa de Entrar sem ter Saído
		28 = Acesso Negado na Tentativa de Sair sem ter Entrado
		48 = Tentativa de Sair do Nível1 sem ter Saído do Nível2

		49 = Tentativa de Entrar em Nível2 sem ter Entrado Nível1
B [1]	ActualLevel	//Nível atual da pessoa 1 a 99
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedAntiPassback rav = new
        ResponseAccessDeniedAntiPassback();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;

        // PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                    rts.EventStructure = rav;
                    rts.DeviceID = args.Msg.DeviceID;
                    rts.AppConnectionStatus = 2;
                    rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                    rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                    rts.UserMessage = new Byte[32];

        DFS.Execute(rts);
        }
```

9.8.19. Acesso Negado por Afastamento

Função:

int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
I[1]	EventID	// 6=Acesso negado por afastamento
D[10]	RemovalBegin	//Data Inicio do Afastamento dd/mm/aaaa
D[10]	RemovalEnd	//Data Final do Afastamento dd/mm/aaaa
B [1]	ActualLevel	//Nível atual da pessoa 1 a 99

I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedRemoval rav = new ResponseAccessDeniedRemoval();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;
        rav.RemovalBegin = Convert.ToDateTime("06:00:00 10/07/2012");
        rav.RemovalEnd = Convert.ToDateTime("07:00:00 10/07/2012");


        // PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                    rts.EventStructure = rav;
                    rts.DeviceID = args.Msg.DeviceID;
                    rts.AppConnectionStatus = 2;
                    rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                    rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                    rts.UserMessage = new Byte[32];


        DFS.Execute(rts);

        }
```

9.8.20. Acesso Negado Crachá Não Encontrado

Função:

int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
B[1]	EventID	// 17=Acesso negado Crachá não encontrado

I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedCardNotFound rav = new
        ResponseAccessDeniedCardNotFound();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;

        // PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                    rts.EventStructure = rav;
                    rts.DeviceID = args.Msg.DeviceID;
                    rts.AppConnectionStatus = 2;
                    rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                    rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                    rts.UserMessage = new Byte[32];


        DFS.Execute(rts);

        }
```

9.8.21. Acesso Negado Cartão Não Registrado

Função:

int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
B[1]	EventID	// 33=Acesso negado Leitora não cadastrada
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso

B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedCardNotRegistered rav = new
        ResponseAccessDeniedCardNotRegistered();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;

        // PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                    rts.EventStructure = rav;
                    rts.DeviceID = args.Msg.DeviceID;
                    rts.AppConnectionStatus = 2;
                    rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                    rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                    rts.UserMessage = new Byte[32];


        DFS.Execute(rts);

        }
```

9.8.22. Acesso Negado Tempo Mínimo de Permanência

Função:

int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
B[1]	EventID	// 41=Acesso negado tempo mínimo permanência
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedMinTime rav = new ResponseAccessDeniedMinTime();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;
        //PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                    rts.EventStructure = rav;
                    rts.DeviceID = args.Msg.DeviceID;
                    rts.AppConnectionStatus = 2;
                    rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                    rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                    rts.UserMessage = new Byte[32];


        DFS.Execute(rts);

        }
```

9.8.23. Acesso Negado Acompanhante

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
B[1]	EventID	// 36=Acesso negado acompanhante
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;
```

```
// Cabeçalho da resposta
ResponseAccessDeniedAccompany rav = new ResponseAccessDeniedAccompany();
                rav.DeviceID = args.Msg.DeviceID;
                rav.ActualLevel = 0;

//PayLoad da Resposta
ResponsePersonStructure rts = new ResponsePersonStructure();
                rts.EventStructure = rav;
                rts.DeviceID = args.Msg.DeviceID;
                rts.AppConnectionStatus = 2;
                rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                rts.UserMessage = new Byte[32];


DFS.Execute(rts);

}
```

9.8.24. Acesso Negado Autorizador Inválido

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
B[1]	EventID	// 46=Acesso negado Autorizador invalido
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedinvalidAuthorizer rav = new
        ResponseAccessDeniedinvalidAuthorizer();
                rav.DeviceID = args.Msg.DeviceID;
                rav.ActualLevel = 0;
```

```
//PayLoad da Resposta
ResponsePersonStructure rts = new ResponsePersonStructure();
                rts.EventStructure = rav;
                rts.DeviceID = args.Msg.DeviceID;
                rts.AppConnectionStatus = 2;
                rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                rts.UserMessage = new Byte[32];


DFS.Execute(rts);

}
```

9.8.25. Acesso Negado por Filial Bloqueada

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
B[1]	EventID	// 54=Acesso negado Filial Bloqueada
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedSubsidiary rav = new ResponseAccessDeniedSubsidiary();
                rav.DeviceID = args.Msg.DeviceID;
                rav.ActualLevel = 0;

        //PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                rts.EventStructure = rav;
                rts.DeviceID = args.Msg.DeviceID;
                rts.AppConnectionStatus = 2;
                rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                rts.UserMessage = new Byte[32];
```

```
DFS.Execute(rts);

}
```

9.8.26. Acesso Negado - Id de Uso de Crachá Bloqueado

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
B[1]	EventID	// 55=Acesso negado Uso Cracha Bloqueado
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON){

        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedCardUseID rav = new ResponseAccessDeniedCardUseID();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;

        //PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                    rts.EventStructure = rav;
                    rts.DeviceID = args.Msg.DeviceID;
                    rts.AppConnectionStatus = 2;
                    rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                    rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                    rts.UserMessage = new Byte[32];


        DFS.Execute(rts);

        }
```

9.8.27. Acesso Negado - Crachá com Cópia

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I[1]	ResponseType	// 1=crachá, 2=pessoa, 3=mensagem
B[1]	EventID	// 56=Acesso negado Cracha Invalido
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [5]	PersonPhisycalId	//Identificador físico do cartão da pessoa
B [5]	PersonLogicalId	// Identificadorlógico do cartão da pessoa (Número da matrícula)
D	TimestampRequest	//Data do acesso
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora
S[32]	UserMessage	//Mensagem pesonalizada a ser exibida (2 linhas de 16 posições cada), zeros assume default indicada EventID

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_ACCESS_REQUEST_PERSON)
        {
        //Para pegar o id da pessoa – matricula ou identificador
        Byte[] personID = (Byte[])((AccessRequestPerson)args.Msg).PersonId;

        // Cabeçalho da resposta
        ResponseAccessDeniedCardCopy rav = new ResponseAccessDeniedCardCopy();
                    rav.DeviceID = args.Msg.DeviceID;
                    rav.ActualLevel = 0;

        //PayLoad da Resposta
        ResponsePersonStructure rts = new ResponsePersonStructure();
                    rts.EventStructure = rav;
                    rts.DeviceID = args.Msg.DeviceID;
                    rts.AppConnectionStatus = 2;
                    rts.PersonPhisycalId = new Byte[] {0x01,0x02,0x03,0x04,0x05};
                    rts.PersonLogicalId = new Byte[] {0x01,0x02,0x03,0x04,0x05 };
                    rts.UserMessage = new Byte[32];


        DFS.Execute(rts);
```

9.8.28. Requisição de Biometria

Descrição:
Esta mensagem é utilizada quando a biometria pela primeira vez não esta salva no sensor, então é solicitado ao DFS/APP o template e depois de validado é salvo no sensor.

Função:
int Execute(rts);

Parâmetros:

I [1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
B[1]	BioConfLevel	//Nível de conferencia biométrico da leitora, retornar de 1-100
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [1]	CardTech	// Categoria: 00 - DESCONHECIDA 01 – BARRAS 02 – PROXIMIDADE 03 – SMARTCARD
B[2048]	Template	//Template
B [1]	TamplateFactory	//Fabricante do template 1–Sagem 2–Geomok 3–IR 4–LG 5–Suprema
B [23]	PersonId	//Identificador da pessoa que possui o cartão. Obs:Caso desejar coletar biometria utilizando o teclado : Deve conter apenas números; O identificador terá apenas 16 caracteres, preenchendo a esquerda com 7 espaços(" ") .Ex : " 0000000123456789" .

Exemplo C#

```
            if (args.Msg.MsgType == MessageType.MSG_TEMPLATE_REQUEST)
            {
             Byte[] bt = (Byte[])((DataRequestTemplate)args.Msg).PersonId;
           ResponseDataTemplate rav = new ResponseDataTemplate();
 TemplateListPersonTemplate[] tmpLstPerTemp = new TemplateListPersonTemplate[2];
        TemplateListPersonCard[] tmpLstPersCard = new TemplateListPersonCard[1];
          TemplateListPerson tmpLstPer = new TemplateListPerson();

               //biometria 1
          tmpLstPerTemp[0] = new TemplateListPersonTemplate();
          tmpLstPerTemp[0].DeviceID = 1;
          tmpLstPerTemp[0].TamplateFactory = 1;
          tmpLstPerTemp[0].Template = new byte[500];
               //biometria 2
          tmpLstPerTemp[1] = new TemplateListPersonTemplate();
          tmpLstPerTemp[1].DeviceID = 1;
          tmpLstPerTemp[1].TamplateFactory = 1;
         tmpLstPerTemp[1].Template = new byte[500];
```

```
            //Cartão 1
            tmpLstPersCard[0] = new TemplateListPersonCard();
            tmpLstPersCard[0].CardID = new Byte[] { 0x00, 0x6A, 0xA9, 0x87, 0x5E };
            tmpLstPersCard[0].DeviceID = 1;
            tmpLstPersCard[0].CardTec = 3;

            //  tmpLstPer = new TemplateListPerson();
            tmpLstPer.BioConfLevel = 75;
            tmpLstPer.DeviceID = 1;
            tmpLstPer.Card = tmpLstPersCard;
            tmpLstPer.Template = tmpLstPerTemp;
            tmpLstPer.PersonID =
        Encoding.ASCII.GetBytes("0000000123456789".PadLeft(23));
            rav.Person = tmpLstPer;
            rav.AppConnectionStatus = 2;
            rav.DeviceID = 1;

            DFS.Execute(rav);
    }
```

9.8.29. Requisição de Dados de Pessoa Relacionada a Cartão

Descrição:
Esta mensagem é utilizada quando do cadastramento de biometria local o FWR não localiza a pessoa associada ao cartão, por isso são solicitados os dados da pessoa.

Função:
int Execute(rts);

Parâmetros:

I [1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [1]	CardTech	// Categoria: 00 - DESCONHECIDA 01 – BARRAS 02 – PROXIMIDADE 03 – SMARTCARD
B[5]	CardID	//Id do crachá no formato Hexadecimal
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora

Exemplo C#

```
   if (args.Msg.MsgType == MessageType.MSG_DATA_REQUEST_CARD) {

    Byte[] cardId = (Byte[])((DataRequestCard)args.Msg).CardId;

ResponseDataCard rav = new ResponseDataCard();
TemplateListPersonCard[] tmpLstPersCard = new TemplateListPersonCard[1];

          //Cartão 1
          tmpLstPersCard[0] = new TemplateListPersonCard();
          tmpLstPersCard[0].CardID = new Byte[] { 0x00, 0x6A, 0xA9, 0x87, 0x5E };
          tmpLstPersCard[0].DeviceID = 1;
          tmpLstPersCard[0].CardTec = 3;
```

```
rav.Card = tmpLstPersCard;
rav.AppConnectionStatus = 2;
rav.DeviceID = 1;
DFS.Execute(rav);
  }
```

9.8.30. Requisição de Dados de Pessoa Relacionada a Identificador

Descrição:
Esta mensagem é utilizada quando do cadastramento de biometria local o FWR não localiza a pessoa associada a um identificador, por isso é solicitados os dados da pessoa.

Função:
int Execute(rts);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
B [1]	CardTech	// Categoria: 00 - DESCONHECIDA 01 – BARRAS 02 – PROXIMIDADE 03 – SMARTCARD
B[5]	CardID	//Id do crachá no formato Hexadecimal
B[1]	Function	//Tecla pressionada.
B[1]	ReaderID	//Número da leitora

Exemplo C#

```
   if (args.Msg.MsgType == MessageType.MSG_DATA_REQUEST_PERSON) {

Byte[] bt = (Byte[])((DataRequestPerson)args.Msg).PersonId;

ResponseDataCard rav = new ResponseDataCard();
TemplateListPersonCard[] tmpLstPersCard = new TemplateListPersonCard[1];

          //Cartão 1
          tmpLstPersCard[0] = new TemplateListPersonCard();
          tmpLstPersCard[0].CardID = new Byte[] { 0x00, 0x6A, 0xA9, 0x87, 0x5E };
          tmpLstPersCard[0].DeviceID = 1;
          tmpLstPersCard[0].CardTec = 3;

          rav.Card = tmpLstPersCard;
          rav.AppConnectionStatus = 2;
          rav.DeviceID = 1;

          DFS.Execute(rav);

            }
```

9.8.31. Resposta Mensagem

Descrição:
Esta mensagem é utilizada quando há a necessidade de enviar uma mensagem a controladora, note que não há acionamentos.

Função:
int Execute(rav);

Parâmetros:

I[1]	AppConnectionStatus	// Estado da comunicação, 1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando

Exemplo C#

```
if (args.Msg.MsgType == MessageType.MSG_DATA_REQUEST_PERSON) {

Byte[] bt = (Byte[])((DataRequestPerson)args.Msg).PersonId;

          ResponseScreenMessage rts = new ResponseScreenMessage();
          rts.DeviceID = args.Msg.DeviceID;
          rts.AppConnectionStatus = 2;
          rts.ResponseType = 3;
          rts.MessageId = 1;
          rts.ScreenMessage = Encoding.ASCII.GetBytes("Mensagem ".PadRight(32));

                    this.DFS.Execute(rts);

}
```

9.9. Eventos

Definidos como eventos, qualquer comportamento monitorado pelo equipamento.
Seguem as sessões a seguir para eventos de equipamentos do tipo Acesso e tipo Relógio.

9.9.1. Eventos de Acesso

Os equipamentos com Firmware Digicon além dos Eventos de conexão tratados pelo DFS e requisição de acesso 12(0x0C) tratados pela aplicação, gera eventos com o identificador 10(0x0A) tratados pela aplicação. Esta sessão trata exclusivamente de como a Aplicação deverá tartar tais eventos que passam atravez do DFS com tratamentos específicos em alguns campos. Sempre que a aplicação recebe esses eventos devolve em resposta o handshake ACK. Enquanto o Firmware não receber ACK, continuará periodicamente enviando o evento à aplicação.

Parâmetros:

I [1]	CommStatus	// Estado da comunicação,1=Offline, 2=Online
I [9]	DeviceID	// ID do dispositivo para onde deverá ser enviado o comando
I [9]	SeqCmd	// Sequencial do comando
D	EventInitDateTime	// Data e Hora Inicial “dd/mm/aaaa hh:mm:ss”
D	EventEndDateTime	// Data e Hora final “dd/mm/aaaa hh:mm:ss”
I [1]	IdentificationType	// Tipo de identificador , 1 = cartão; 2 = id da pessoa.

B [23]	IdentificationData	// Identificador do cartão ou da pessoa, conforme campo acima.Quando cartão, os primeiros 5 bytes representam o identificador, e o restante dos bytes devem ser lidos e descartados, virão zerados
I[1]	EventType	//Tipo do evento 0=offline,1=online
I[1]	SummerTime	//Horário de verão, 0 = não,1 = sim
I[1]	GMT1	//GMT (-3) = -180
I[1]	AcessDirection	//Direção do acesso,1 = entrada, 2 = saída
I [1]	ReaderID	// Código da leitora
I[1]	TemplateQuantity	//Quantidade de templates registradas.
I	TemplateSize	//Quantidade debytes do template.
B[]	Template	//Template

I[1]	TemplateVendor	// Fabricante do template
		1=Sagem,
		2=Geomok,
		3=IR,
		4=LG,
		5=Suprema

I	MemoryFreeSpace	//Memória livre
I	InputId	//Id da entrada
I	InputStatus	//Estatus da entrada

Exemplo C#

```
 if (args.Msg.MsgType == MessageType.MSG_EVENT)
   {
String msg="";
if ((EventMessageType)((EventMessage)args.Msg).EventID > 0 )
  {
"cod evento " + ((EventMessage)args.Msg).EventID.ToString() +
"\nDeviceID.......= "  + ((EventMessage)args.Msg).DeviceID.ToString() +
"\nData   =" + ((EventMessage)args.Msg).Timestamp.ToString() +
"\nDireção =" + ((EventMessage)args.Msg).AcessDirection.ToString() +
"\nVerão   =" + ((EventMessage)args.Msg).SummerTime.ToString() +
"\ntipo=" + ((EventMessage)args.Msg).IdentificationType.ToString() +
"\nid =" +
       BitConverter.ToString(((EventMessage)args.Msg).IdentificationData).Replace
       ("-", "") +
"\nGmt     =" + ((EventMessage)args.Msg).GMT1.ToString() +
"\nReader  =" + ((EventMessage)args.Msg).ReaderID.ToString() ;

//caso acesso permitido
  if ((EventMessageType)((EventMessage)args.Msg).EventID ==
         EventMessageType.MSG_EVENT_ACCESS_GRANTED ||
(EventMessageType)((EventMessage)args.Msg).EventID ==
       EventMessageType.MSG_EVENT_ACCESS_GRANTED_CHEAT ||
(EventMessageType)((EventMessage)args.Msg).EventID ==
       EventMessageType.MSG_EVENT_ACCESS_GRANTED_COERCION ||
(EventMessageType)((EventMessage)args.Msg).EventID ==
       EventMessageType.MSG_EVENT_ACCESS_GRANTED_FRISK ||
(EventMessageType)((EventMessage)args.Msg).EventID ==
       EventMessageType.MSG_EVENT_ACCESS_GRANTED_MASTER_CARD ||
(EventMessageType)((EventMessage)args.Msg).EventID ==
       EventMessageType.MSG_EVENT_ACCESS_GRANTED_OUT_REPOSE ||
(EventMessageType)((EventMessage)args.Msg).EventID ==
       EventMessageType.MSG_EVENT_ACCESS_VALID_ACCOMPANY)
{
msg += "\n\nTecla pressionada........" +
((EventAccessGranted)args.Msg).KeyPressed.ToString() +
```

```
 "\nMsgQuantity" + ((EventAccessGrantedBase)args.Msg).MsgQuantity.ToString() +
"\nAcessCredits" + ((EventAccessGrantedBase)args.Msg).AcessCredits.ToString() +
"\nCardLevel " + ((EventAccessGrantedBase)args.Msg).CardLevel.ToString();
  }

//acesso negado
  else if ((EventMessageType)((EventMessage)args.Msg).EventID ==
EventMessageType.MSG_EVENT_ACCESS_DENIED_LEVEL) {
    msg += "\nTipo de fluxo ........ " +
((EventAccessDeniedLevel)args.Msg).FlowTipe.ToString()+
                                        "\nNível do cartão ........ " +
((EventAccessDeniedLevel)args.Msg).CardLevel.ToString();
                                }

else if ((EventMessageType)((EventMessage)args.Msg).EventID ==
EventMessageType.MSG_EVENT_ACCESS_DENIED_ANTIPASSBACK){
  msg += "\nTipo de fluxo ........ " +
((EventAccessDeniedAntiPassback)args.Msg).FlowTipe.ToString() + "\nNível do
cartão ........ " +
((EventAccessDeniedAntiPassback)args.Msg).CardLevel.ToString();
                                }
 else if ((EventMessageType)((EventMessage)args.Msg).EventID ==
EventMessageType.MSG_EVENT_ACCESS_DENIED_REMOVAL){
  msg += "\nTipo de fluxo ........ " +
((EventAccessDeniedRemoval)args.Msg).FlowTipe.ToString()+
    "\nNível do cartão ........ " +
((EventAccessDeniedRemoval)args.Msg).CardLevel.ToString();
                                }

//controle de memória
else if ((EventMessageType)((EventMessage)args.Msg).EventID ==
EventMessageType.MSG_EVENT_MEMORY_CONTROL)
{
msg += "MemoryFreeSpace" +
((EventMemoryControl)args.Msg).MemoryFreeSpace.ToString();
}

//caso firmware inicializado
else if ((EventMessageType)((EventMessage)args.Msg).EventID ==
EventMessageType.MSG_EVENT_FIRMWARE_STARTED)
{
msg += "\nMemoria livre " +
((EventFirmwareInitialized)args.Msg).MemoryFreeSpace.ToString();
}

//Caso seja retorno de template registrada
else if ((EventMessageType)((EventMessage)args.Msg).EventID ==
EventMessageType.MSG_EVENT_TEMPLATE_REGISTERED)
{
msg += "\nTotal Templates = " +
((EventTemplateRegister)args.Msg).TemplateQuantity.ToString() + "\n";
msg += "\nSize = " + ((EventTemplateRegister)args.Msg).Payload .ToString() +
"\n";
for (int i = 0; i < ((EventTemplateRegister)args.Msg).TemplateQuantity; i++)
  {
msg += "\nTemplate " + i.ToString() +
  "\nTemplateVendor = " +
((EventTemplateRegister)args.Msg).Template[i].TemplateVendor.ToString() +
"\nTemplateSize = " +
((EventTemplateRegister)args.Msg).Template[i].TemplateSize.ToString() +
"\nTemplate \n";
```

```
msg +=
ArrayUtilities.GetByteArrayAsHexa(((EventTemplateRegister)args.Msg).Template[i].T
emplate) + "\n";
  }
}
//caso leitora biométrica não encontrada
else if ((EventMessageType)((EventMessage)args.Msg).EventID ==
EventMessageType.MSG_EVENT_BIOMETRIC_READER_NOT_FOUND)
{
msg += "Device IP ....." +
((EventUnregisteredBioReader)args.Msg).DeviceIp[0].ToString() + "." +
((EventUnregisteredBioReader)args.Msg).DeviceIp[1].ToString() + "." +
((EventUnregisteredBioReader)args.Msg).DeviceIp[2].ToString() + "." +
((EventUnregisteredBioReader)args.Msg).DeviceIp[3].ToString() +
"\nReader Location ....." +
((EventUnregisteredBioReader)args.Msg).ReaderLocation.ToString();
                    }
Console.WriteLine(msg);
            }
}
```

9.9.2. Eventos de Ponto

Os eventos de equipamento do tipo ponto são requisitados através dos comandos **RequestBackupLog** e **RequestLog** .

Estarão no array dos objetos nos quais são requisitados.

Abaixo os eventos, suas descrições e propriedades do enumerado **`EventClockType`**, no java **`EventClockTypeJava`**:

Evento	**Descrição**
EventCollectionOfTheLast24H	Coleta das ultimas 24Hs de marcações
EventCompartmentPaperClose	Compartimento de papel fechado
EventCompartmentPaperOpen	Compartimento de papel aberto
EventCutterIsBlock	Guilhotina travada
EventDeviceInfriged	Equipamento violado
EventDeviceUnblocked	Equipamento desbloqueado
EventEnrollEvent	Cadastro de cartÃµes ou biometrias
EventExchangePaper	Efetuada a troca de papel
EventLowLevelOfPaper	Nível baixo de papel (bobina com 5 % do tamanho)
EventMrpFull	Memória MRP cheia
EventMrpLowLevel	Nível baixo de memoria MRP
EventPendriveConnectedCommPort	Pendrive conectado na porta de comunicação
EventPendriveConnectedFiscalPort	Pendrive conectado na porta fiscal
EventPrinterFailed	Falha na impressora
EventPrinterOutOfPaper	Impressora sem papel
EventRegisteredPoint	Ponto registrado
EventRepInseredOnTheWall	REP foi afixado na parede
EventRepIsLoged	Indica que o REP foi logado na rede
EventRepReadyToSendBlock	Indica que o REP está pronto para enviar um bloco de marcações

EventRepReadyToSendLastBlock	Indica que o REP está pronto para enviar o último bloco de marcações coletados
EventRemovedFromTheWall	REP foi removido da parede
EventThereIsNotComm	Indica que não há conexão de rede entre o REP e o servidor
EventNoDataFileLog	Indica que não existe eventos na MRP
EventGetEventsUsingPendrive	Coleta de eventos através do pendrive.
EventGetDataUsingPendrive	Coleta de banco de dados através do pendrive

10. Problemas, compreendendo e resolvendo

Mesmo com toda atenção e preocupação constante com nossos produtos, não estamos imunes de erros ou problemas.

Nosso processo de qualidade prevê melhoria contínua nos processos para desenvolvimento de produtose serviços, neste sentido estamos abertos a críticas e sugestões.

Estamos trabalhando compilando informações sobre análise de erros e situações encontradas pelos usuários e em breve farão parte deste manual.

Acesse no site: www.digicon.com.br

Se preferir enviar e-mail para: suporte.vca@digicon.com.br

11. Glossário

- Java ➔ Linguagem de programação orientada a objeto, desenvolvida na década de 90 por uma equipe de programadores chefiada por James Gosling, na empresa Sun Microsystems.
- JVM➔Java Virtual Machine, responsável pela interpretação do código Java.
- JNI ➔ Java Natural Interface, conjunto de ferramentas e bibliotecas para integração entre o C# e Java.
- JNI4NET ➔ Biblioteca de integração padrão JNI com C# ddo DFS.
- J2EE➔Java Platform Enterprise Edition ou Java Edição Empresarial é uma plataforma de programaçãopara servidores na linguagemde programaçãoJava.
- Windows➔ Marca registrada de sistema operacional da Microsoft Corporation. (www.microsoft.com).
- Middleware ➔ Em informática é um mediador, composto de hardware e/ou software, comumente utilizado para mover ou transportar informações e dados entre programas de diferentes protocolos de comunicação e plataformas.
- MCA ➔ Sigla Digicon que significa: Módulo Controlador de Acesso, principal placa de acesso baseada no processador Motorola Power PC XPC850.
- MCANet ➔ Controladora baseada no processador ARM 9.
- MRA➔Módulo Remoto de Acesso Digicon, leitora inteligente que trabalha com protocolo RS 485 usando como controladora master uma MCANet.
- SDK➔Software Development Kit, conjunto de bibliotecas e padrões de protocolo para desenvolvimento de aplicativos para integração.
- FTP➔ File Transfer Protocol, protocolo para transferência de arquivos.
- TCP➔ Transport Control Protocol, camada de transporte do protocolo TCP/IP.
- DLL ➔Dynamic Link Library, biblioteca de vínculo dinâmico, desenvolvida inicialmente pela Microsoft para o conceito de biblioteca compartilhada.
- API ➔Application Programming Interface, bibliotecas ou camadas intermediárias, normalmente fazem parte do SDK.
- MRP ➔ Sigla referente a portaria 1.510 que significa, Memória do Registrador de Ponto.