LAMPIRAN A

Listing Program Server dan Client
Lampiran

PROGRAM SERVER GERBANG TOL MASUK

Private Sub bntConnect_Click()
On Error GoTo t
sock1.Close
sock1.RemoteHost = txtIP
sock1.RemotePort = txtPort
sock1.Connect
Exit Sub
t:
MsgBox "Error : " & Err.Description, vbCritical
End Sub

Private Sub bntExit_Click()
End
End Sub

Private Sub bntSend_Click()
On Error GoTo t
sock1.SendData txtSend
txtLog = txtLog & "Client : " & txtSend & vbCrLf
txtSend = ""
Exit Sub
t:
MsgBox "Error : " & Err.Description
sock1.Close
End Sub

Private Sub DEC_Change()
DECHEX.Text = Hex(DEC.Text)
End Sub
Private Sub Dec_val_Click()
    txtLog.Text = ""
    decval.Text = ""
    Timer5.Enabled = True
    On Error GoTo t
    Do While (Len(DECHEX.Text) < 8)
        DECHEX.Text = "0" & DECHEX.Text
    Loop
    sock1.SendData "-" & Block2.Text & DECHEX.Text
    Exit Sub
    t:
    MsgBox "Error : " & Err.Description
    sock1_Close
    End Sub

Private Sub Form_Load()
    ClientFrm.Height = 1335
    ClientFrm.Width = 3765
    End Sub

Private Sub ID_capture_Click()
    txtLog.Text = ""
    Result_2.Text = ""
    Timer2.Enabled = True
    On Error GoTo t
    sock1.SendData "s"
    Exit Sub
    t:
    MsgBox "Error : " & Err.Description
Private Sub Login_Click()

    Result_3.Text = ""
    txtLog.Text = ""
    Timer3.Enabled = True
    On Error GoTo t
    sock1.SendData "l" & block.Text & "AA" & KEY
    Exit Sub
    t:
    MsgBox "Error : " & Err.Description
    sock1_Close
    End Sub

Private Sub Read_Click()

    txtLog.Text = ""
    RD.Text = ""
    Timer6.Enabled = True
    On Error GoTo t
    sock1.SendData "r" & Block3.Text
    Exit Sub
    t:
    MsgBox "Error : " & Err.Description
    sock1_Close
    End Sub

Private Sub Read_val_Click()

    Timer14.Enabled = True
txtLog.Text = ""
RV.Text = ""
Timer4.Enabled = True
On Error GoTo t
sock1.SendData "rv" & Block2.Text
Exit Sub
t:
MsgBox "Error : " & Err.Description
sock1_Close

End Sub
Private Sub RESET_Click()
txtLog.Text = ""
Result_1.Text = ""
Timer1.Enabled = True
On Error GoTo t
sock1.SendData "x"
Exit Sub
t:
MsgBox "Error : " & Err.Description
sock1_Close
End Sub

Private Sub sock1_Close()
sock1.Close
txtLog = txtLog & "*** Disconnected" & vbCrLf
Timer8.Enabled = True
ClientFrm.Height = 1335
ClientFrm.Width = 4800
RD.Text = ""
Result_1.Text = ""
Result_2.Text = ""
Result_3.Text = ""
Text_3.Text = ""

Timer1.Enabled = True
Timer2.Enabled = True
Timer3.Enabled = True
Timer4.Enabled = True
Timer5.Enabled = True
Timer6.Enabled = True
Timer7.Enabled = True
Timer8.Enabled = True
Timer9.Enabled = True
Timer10.Enabled = True
Timer11.Enabled = True
Timer12.Enabled = True
Timer13.Enabled = True
Timer14.Enabled = True

End Sub
Private Sub sock1_Connect()
    txtLog = "Connected to " & sock1.RemoteHostIP & vbCrLf
    Timer8.Enabled = False
    ClientFrm.Height = 4200
    ClientFrm.Width = 3765
    End Sub
Private Sub sock1_DataArrival(Val bytesTotal As Long)
Dim dat As String
sock1.GetData dat, vbString
If Timer1.Enabled = True Then Result_1.Text = Result_1.Text + dat
If Timer2.Enabled = True Then Result_2.Text = Result_2.Text + dat
If Timer3.Enabled = True Then Result_3.Text = Result_3.Text + dat
If Timer4.Enabled = True Then RV.Text = RV.Text + dat
If Timer5.Enabled = True Then decval.Text = decval.Text + dat
If Timer6.Enabled = True Then RD.Text = RD.Text + dat
If Timer7.Enabled = True Then WRT.Text = WRT.Text + dat

txtLog = txtLog & dat & vbCrLf
End Sub

Private Sub sock1_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean)

txtLog = txtLog & "*** Error : " & Description & vbCrLf
sock1.Close
End Sub

Private Sub Timer1_Timer()
Timer1.Enabled = False
End Sub

Private Sub Timer10_Timer()
If Timer8.Enabled = False Then
Call RESET_Click
Timer10.Enabled = False
End If
Private Sub Timer11_Timer()
If Timer8.Enabled = False And Len(Result_2.Text) < 6 Then
Call ID_capture_Click
Timer11.Enabled = False
End If
End Sub

Private Sub Timer12_Timer()
If Timer8.Enabled = False And Len(Result_3.Text) < 6 Then
Call Login_Click
Timer12.Enabled = False
End If
End Sub

Private Sub Timer13_Timer()
If Timer8.Enabled = False And Len(Text3.Text) < 6 Then
Call Read_val_Click
Timer13.Enabled = False
End If
End Sub

Private Sub Timer14_Timer()
If Val(Text3.Text) > 5000 Then
Call Write_Click
End If
Timer14.Enabled = False
Private Sub Timer2_Timer()
    Timer2.Enabled = False
End Sub

Private Sub Timer3_Timer()
    Timer3.Enabled = False
End Sub

Private Sub Timer4_Timer()
    Timer4.Enabled = False
End Sub

Private Sub Timer5_Timer()
    Timer5.Enabled = False
End Sub

Private Sub Timer6_Timer()
    Timer6.Enabled = False
End Sub

Private Sub Timer7_Timer()
    Timer7.Enabled = False
End Sub

Private Sub Timer8_Timer()
    Call bntConnect_Click
End Sub

Private Sub Timer9_Timer()
    Text3.Text = (Val(Mid(RV.Text, 1, 1)) * 268435456) + (Val(Mid(RV.Text, 2, 1)) * 16777216) + (Val(Mid(RV.Text, 3, 1)) * 1048576) + (Val(Mid(RV.Text, 4, 1)) * 65536) + (Val(Mid(RV.Text, 5, 1)) * 4096) + (Val(Mid(RV.Text, 6, 1)) * 256) + (Val(Mid(RV.Text, 7, 1)) * 16) + (Val(Mid(RV.Text, 8, 1)))

    For i = 1 To 8
        Text2.Text = 0
    Next i
Select Case Mid(RV.Text, i, 1)
Case "A"
    Text2.Text = 10
Case "B"
    Text2.Text = 11
Case "C"
    Text2.Text = 12
Case "D"
    Text2.Text = 13
Case "E"
    Text2.Text = 14
Case "F"
    Text2.Text = 15
End Select
Select Case i
Case 1
Case 2
Case 3
Case 4
Case 5
Case 6
Case 7

Universitas Kristen Maranatha
Case 8


End Select

Next i

Text4.Text = (Val(Mid(decval.Text, 1, 1)) * 268435456) + (Val(Mid(decval.Text, 2, 1)) * 16777216) + (Val(Mid(decval.Text, 3, 1)) * 1048576) + (Val(Mid(decval.Text, 4, 1)) * 65536) + (Val(Mid(decval.Text, 5, 1)) * 4096) + (Val(Mid(decval.Text, 6, 1)) * 256) + (Val(Mid(decval.Text, 7, 1)) * 16) + (Val(Mid(decval.Text, 8, 1)))

For i = 1 To 8

Text1.Text = 0

Select Case Mid(decval.Text, i, 1)

Case "A"

Text1.Text = 10

Case "B"

Text1.Text = 11

Case "C"

Text1.Text = 12

Case "D"

Text1.Text = 13

Case "E"

Text1.Text = 14

Case "F"

Text1.Text = 15

End Select

Select Case i

Case 1

Case 2

Case 3

Case 4

Case 5

Case 6

Case 7

Case 8

End Select
Next i

End Sub

Private Sub Write_Click()

wrtval.Text = ""

i = Len(Combo1.Text)
For y = 1 To i
wrtval.Text = Hex(Asc(Right(Combo1.Text, y))) + wrtval.Text
Next y
Do While (Len(wrtval.Text) < 32)
    wrtval.Text = "0" & wrtval.Text
Loop

txtLog.Text = ""
WRT.Text = ""
Timer7.Enabled = True
On Error GoTo t
sock1.SendData "w" & Block3.Text & wrtval.Text
Exit Sub

End Sub

PROGRAM SERVER GERBANG TOL KELUAR PADALARANG

Private Sub bntConnect_Click()
On Error GoTo t
sock1.Close
sock1.RemoteHost = txtIP
sock1.RemotePort = txtPort
sock1.Connect
Exit Sub

End Sub
Private Sub bntExit_Click()
    End
End Sub

Private Sub bntSend_Click()
    On Error GoTo t
sock1.SendData txtSend
txtLog = txtLog & "Client : " & txtSend & vbCrLf
txtSend = ""
Exit Sub
t:
MsgBox "Error : " & Err.Description
sock1_Close
End Sub

Private Sub DEC_Change()
DECHEX.Text = Hex(DEC.Text)
End Sub

Private Sub Dec_val_Click()
txtLog.Text = ""
decval.Text = ""
Timer5.Enabled = True
On Error GoTo t
Do While (Len(DECHEX.Text) < 8)
DECHEX.Text = "0" & DECHEX.Text
Loop
sock1.SendData ":-" & Block2.Text & DECHEX.Text
Exit Sub
t:
MsgBox "Error : " & Err.Description
Private Sub Form_Load()

Picture2.Left = 14100

ClientFrm.Height = 1335
ClientFrm.Width = 4800

End Sub

Private Sub ID_capture_Click()

txtLog.Text = ""
Result_2.Text = ""
Timer2.Enabled = True

On Error GoTo t

sock1.SendData "s"

Exit Sub

t:

MsgBox "Error : " & Err.Description

sock1_Close

End Sub

Private Sub Login_Click()

Result_3.Text = ""

txtLog.Text = ""

Timer3.Enabled = True

On Error GoTo t

sock1.SendData "l" & block.Text & "AA" & KEY

Exit Sub

t:

MsgBox "Error : " & Err.Description
sock1.Close
End Sub

Private Sub Read_Click()
  Timer15.Enabled = True

  txtLog.Text = ""
  RD.Text = ""
  Timer6.Enabled = True
  On Error GoTo t
  sock1.SendData "r" & Block3.Text
  Exit Sub
  t:
  MsgBox "Error : " & Err.Description
  sock1.Close
  End Sub

Private Sub Read_val_Click()
  txtLog.Text = ""
  RV.Text = ""
  Timer4.Enabled = True
  On Error GoTo t
  sock1.SendData "rv" & Block2.Text
  Exit Sub
  t:
  MsgBox "Error : " & Err.Description
  sock1.Close
  End Sub

Private Sub RESET_Click()
  txtLog.Text = ""

Universitas Kristen Maranatha
Result_1.Text = ""
Timer1.Enabled = True
On Error GoTo t
sock1.SendData "x"
Exit Sub
t:
MsgBox "Error : " & Err.Description
sock1.Close
End Sub
Private Sub sock1_Close()
sock1.Close
txtLog = txtLog & "*** Disconnected" & vbCrLf
Timer8.Enabled = True
ClientFrm.Height = 1335
ClientFrm.Width = 4800
End Sub
Private Sub sock1_Connect()
txtLog = "Connected to " & sock1.RemoteHostIP & vbCrLf
Timer8.Enabled = False
ClientFrm.Height = 6045
ClientFrm.Width = 15000
End Sub
Private Sub sock1_DataArrival(ByVal bytesTotal As Long)
Dim dat As String
sock1.GetData dat, vbString
If Timer1.Enabled = True Then Result_1.Text = Result_1.Text + dat
If Timer2.Enabled = True Then Result_2.Text = Result_2.Text + dat
If Timer3.Enabled = True Then Result_3.Text = Result_3.Text + dat
If Timer4.Enabled = True Then RV.Text = RV.Text + dat
If Timer5.Enabled = True Then decval.Text = decval.Text + dat

If Timer6.Enabled = True Then RD.Text = RD.Text + dat

If Timer7.Enabled = True Then WRT.Text = WRT.Text + dat

txtLog = txtLog & dat & vbCrLf

End Sub

Private Sub sock1_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean)

txtLog = txtLog & "*** Error : " & Description & vbCrLf
sock1_Close

End Sub

Private Sub Timer1_Timer()

Timer1.Enabled = False

End Sub

Private Sub Timer10_Timer()
If Timer8.Enabled = False Then
Call RESET_Click

Timer10.Enabled = False

End If

End Sub

Private Sub Timer11_Timer()
If Timer8.Enabled = False Then
Call ID_capture_Click

Timer11.Enabled = False

End If
End Sub

Private Sub Timer12_Timer()
If Timer8.Enabled = False Then
Call Login_Click
Timer12.Enabled = False
End If
End Sub

Private Sub Timer13_Timer()
If Timer8.Enabled = False Then
Call Read_val_Click
Timer13.Enabled = False
End If
End Sub

Private Sub Timer14_Timer()
If Timer8.Enabled = False Then
Call Read_Click
Timer14.Enabled = False
End If
End Sub

Private Sub Timer15_Timer()
If Left(RD.Text, 32) = "0000000000000000000043494D414849" Then
DEC.Text = "1000"
Call Dec_val_Click

asal.Caption = "CIMAHI"
If Left(RD.Text, 32) = "00000000000000000000504153545552" Then
DEC.Text = "2000"
Call Dec_val_Click

asal.Caption = "PASTEUR"
Picture2.Left = -120
End If

Timer15.Enabled = False
End Sub

Private Sub Timer16_Timer()
If Timer8.Enabled = False Then
Call Write_Click
Timer16.Enabled = False
End If
End Sub
Private Sub Timer2_Timer()
Timer2.Enabled = False
End Sub

Private Sub Timer3_Timer()
Timer3.Enabled = False
End Sub

Private Sub Timer4_Timer()
Timer4.Enabled = False
End Sub

Private Sub Timer5_Timer()
Timer5.Enabled = False
End Sub

Private Sub Timer6_Timer()
Timer6.Enabled = False
End Sub

Private Sub Timer7_Timer()
Timer7.Enabled = False
End Sub

Private Sub Timer8_Timer()
Call bntConnect_Click
End Sub

Private Sub Timer9_Timer()
Text3.Text = (Val(Mid(RV.Text, 1, 1)) * 268435456) + (Val(Mid(RV.Text, 2, 1)) * 16777216) + (Val(Mid(RV.Text, 3, 1)) * 1048576) + (Val(Mid(RV.Text, 4, 1)) * 65536) + (Val(Mid(RV.Text, 5, 1)) * 4096) + (Val(Mid(RV.Text, 6, 1)) * 256) + (Val(Mid(RV.Text, 7, 1)) * 16) + (Val(Mid(RV.Text, 8, 1)))
For i = 1 To 8
Text2.Text = 0
Select Case Mid(RV.Text, i, 1)
Case "A"
Text2.Text = 10
Case "B"
Text2.Text = 11
Case "C"
Text2.Text = 12
Case "D"
Text2.Text = 13
Case "E"
Text2.Text = 14
Case "F"
Text2.Text = 15
End Select
Select Case i
Case 1
Case 2
Case 3
Case 4
Case 5
Case 6
Case 7
Case 8
Lampiran

End Select
Next i

Text4.Text = (Val(Mid(decval.Text, 1, 1)) * 268435456) + (Val(Mid(decval.Text, 2, 1)) * 16777216) + (Val(Mid(decval.Text, 3, 1)) * 1048576) + (Val(Mid(decval.Text, 4, 1)) * 65536) + (Val(Mid(decval.Text, 5, 1)) * 4096) + (Val(Mid(decval.Text, 6, 1)) * 256) + (Val(Mid(decval.Text, 7, 1)) * 16) + (Val(Mid(decval.Text, 8, 1)))

For i = 1 To 8
Text1.Text = 0
Select Case Mid(decval.Text, i, 1)
Case "A"
Text1.Text = 10
Case "B"
Text1.Text = 11
Case "C"
Text1.Text = 12
Case "D"
Text1.Text = 13
Case "E"
Text1.Text = 14
Case "F"
Text1.Text = 15
End Select
Select Case i
Case 1
Case 2
Case 3

Case 4

Case 5

Case 6

Case 7

Case 8

End Select
Next i

End Sub
Private Sub Write_Click()

wrtval.Text = ""

i = Len(Text5.Text)
For y = 1 To i
wrtval.Text = Hex(Asc(Right(Text5.Text, y))) + wrtval.Text
Next y

Do While (Len(wrtval.Text) < 32)
wrtval.Text = "0" & wrtval.Text
Loop
txtLog.Text = ""
WRT.Text = ""
Timer7.Enabled = True
On Error GoTo t
sock1.SendData "w" & Block3.Text & wrtval.Text
Exit Sub
t:
MsgBox "Error : " & Err.Description
sock1.Close
End Sub

PROGRAM SERVER GERBANG TOL KELUAR CIMahi
Private Sub bntConnect_Click()
On Error GoTo t
sock1.Close
sock1.RemoteHost = txtIP
sock1.RemotePort = txtPort
sock1.Connect
Exit Sub
t:
MsgBox "Error : " & Err.Description, vbCritical
End Sub
Private Sub bntExit_Click()
End
End Sub
Private Sub bntSend_Click()
On Error GoTo t
sock1.SendData txtSend
txtLog = txtLog & "Client : " & txtSend & vbCrLf
txtSend = ""
Exit Sub

Private Sub DEC_Change()
DECHEX.Text = Hex(DEC.Text)
End Sub

Private Sub Dec_val_Click()
DECHEX.Text = ""
End Sub

Private Sub Form_Load()
Picture2.Left = 3750
ClientFrm.Height = 1335
ClientFrm.Width = 3750
End Sub
Private Sub ID_capture_Click()
txtLog.Text = ""
Result_2.Text = ""
Timer2.Enabled = True
On Error GoTo t
sock1.SendData "s"
Exit Sub
t:
MsgBox "Error : " & Err.Description
sock1.Close
End Sub
Private Sub Login_Click()
Result_3.Text = ""
txtLog.Text = ""
Timer3.Enabled = True
On Error GoTo t
sock1.SendData "l" & block.Text & "AA" & KEY
Exit Sub
t:
MsgBox "Error : " & Err.Description
sock1.Close
End Sub

Private Sub Read_Click()
Timer15.Enabled = True
txtLog.Text = ""
RD.Text = ""
Timer6.Enabled = True
On Error GoTo t
sock1.SendData "r" & Block3.Text
Exit Sub
t:
MsgBox "Error : " & Err.Description
sock1.Close
End Sub
Private Sub Read_val_Click()
    txtLog.Text = ""
    RV.Text = ""
    Timer4.Enabled = True
    On Error GoTo t
    sock1.SendData "rv" & Block2.Text
    Exit Sub
    t:
    MsgBox "Error : " & Err.Description
    sock1.Close
    End Sub
Private Sub RESET_Click()
    txtLog.Text = ""
    Result_1.Text = ""
    Timer1.Enabled = True
    On Error GoTo t
    sock1.SendData "x"
    Exit Sub
    t:
MsgBox "Error : " & Err.Description
sock1_Close
End Sub
Private Sub sock1_Close()
sock1.Close

txtLog = txtLog & "*** Disconnected" & vbCrLf
Timer8.Enabled = True
ClientFrm.Height = 4335
ClientFrm.Width = 3750
End Sub
Private Sub sock1_Connect()

txtLog = "Connected to " & sock1.RemoteHostIP & vbCrLf
Timer8.Enabled = False
ClientFrm.Height = 4260
ClientFrm.Width = 3750
End Sub
Private Sub sock1_DataArrival(ByVal bytesTotal As Long)
Dim dat As String
sock1.GetData dat, vbString
If Timer1.Enabled = True Then Result_1.Text = Result_1.Text + dat
If Timer2.Enabled = True Then Result_2.Text = Result_2.Text + dat
If Timer3.Enabled = True Then Result_3.Text = Result_3.Text + dat
If Timer4.Enabled = True Then RV.Text = RV.Text + dat
If Timer5.Enabled = True Then decval.Text = decval.Text + dat
If Timer6.Enabled = True Then RD.Text = RD.Text + dat
If Timer7.Enabled = True Then WRT.Text = WRT.Text + dat

txtLog = txtLog & dat & vbCrLf
End Sub
Private Sub sock1_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean)
    txtLog = txtLog & "*** Error : " & Description & vbCrLf
    sock1_Close
    End Sub

Private Sub Timer1_Timer()
    Timer1.Enabled = False
    End Sub

Private Sub Timer10_Timer()
    If Timer8.Enabled = False Then
        Call RESET_Click
        Timer10.Enabled = False
    End If
    End Sub

Private Sub Timer11_Timer()
    If Timer8.Enabled = False Then
        Call ID_capture_Click
        Timer11.Enabled = False
    End If
    End Sub

Private Sub Timer12_Timer()
    If Timer8.Enabled = False Then
        Call Login_Click
    End Sub
Timer12.Enabled = False
End If
End Sub

Private Sub Timer13_Timer()
If Timer8.Enabled = False Then
    Call Read_val_Click
    Timer13.Enabled = False
End If
End Sub

Private Sub Timer14_Timer()
If Timer8.Enabled = False Then
    Call Read_Click
    Timer14.Enabled = False
End If
End Sub

Private Sub Timer15_Timer()
If Left(RD.Text, 32) = "000000000000504144414C4152414E47" Then
    DEC.Text = "1000"
    Call Dec_val_Click
    asal.Caption = "PADALARANG"
    Picture2.Left = -120
End If
If Left(RD.Text, 32) = "00000000000000000000504153545552" Then
DEC.Text = "1000"
Call Dec_val_Click
asal.Caption = "PASTEUR"
Picture2.Left = -120
End If

Timer15.Enabled = False
End Sub

Private Sub Timer16_Timer()
If Timer8.Enabled = False Then
Call Write_Click
Timer16.Enabled = False
End If
End Sub

Private Sub Timer2_Timer()
Timer2.Enabled = False
End Sub

Private Sub Timer3_Timer()
Timer3.Enabled = False
End Sub

Private Sub Timer4_Timer()
Timer4.Enabled = False
End Sub

Private Sub Timer5_Timer()
Timer5.Enabled = False
End Sub

Private Sub Timer6_Timer()
Timer6.Enabled = False
End Sub

Private Sub Timer7_Timer()
Timer7.Enabled = False
End Sub

Private Sub Timer8_Timer()
Call bntConnect_Click
End Sub

Private Sub Timer9_Timer()
Text3.Text = (Val(Mid(RV.Text, 1, 1)) * 268435456) + (Val(Mid(RV.Text, 2, 1)) * 16777216) + (Val(Mid(RV.Text, 3, 1)) * 1048576) + (Val(Mid(RV.Text, 4, 1)) * 65536) + (Val(Mid(RV.Text, 5, 1)) * 4096) + (Val(Mid(RV.Text, 6, 1)) * 256) + (Val(Mid(RV.Text, 7, 1)) * 16) + (Val(Mid(RV.Text, 8, 1)))

For i = 1 To 8
Text2.Text = 0
Select Case Mid(RV.Text, i, 1)
Case "A"
Text2.Text = 10
Case "B"
Text2.Text = 11
Case "C"
Lampiran

Text2.Text = 12
Case "D"
Text2.Text = 13
Case "E"
Text2.Text = 14
Case "F"
Text2.Text = 15
End Select
Select Case i
Case 1
Case 2
Case 3
Case 4
Case 5
Case 6
Case 7
Case 8
End Select
Next i
Text4.Text = (Val(Mid(decval.Text, 1, 1)) * 268435456) + (Val(Mid(decval.Text, 2, 1)) * 16777216) + (Val(Mid(decval.Text, 3, 1)) * 1048576) + (Val(Mid(decval.Text, 4, 1)) * 65536) + (Val(Mid(decval.Text, 5, 1)) * 4096) + (Val(Mid(decval.Text, 6, 1)) * 256) + (Val(Mid(decval.Text, 7, 1)) * 16) + (Val(Mid(decval.Text, 8, 1)))

For i = 1 To 8
   Text1.Text = 0
   Select Case Mid(decval.Text, i, 1)
      Case "A"
         Text1.Text = 10
      Case "B"
         Text1.Text = 11
      Case "C"
         Text1.Text = 12
      Case "D"
         Text1.Text = 13
      Case "E"
         Text1.Text = 14
      Case "F"
         Text1.Text = 15
   End Select
   Select Case i
   Case 1
   Case 2
   Case 3
   Case 4
Case 5
Case 6
Case 7
Case 8
End Select
Next i

End Sub
Private Sub Write_Click()

wrtval.Text = ""

i = Len(Text5.Text)
For y = 1 To i
wrtval.Text = Hex(Asc(Right(Text5.Text, y))) + wrtval.Text
Next y

Do While (Len(wrtval.Text) < 32)
wrtval.Text = "0" & wrtval.Text
Loop

txtLog.Text = ""
WRT.Text = ""
Timer7.Enabled = True
On Error GoTo t
sock1.SendData "w" & Block3.Text & wrtval.Text
Exit Sub
t:
MsgBox "Error : " & Err.Description
sock1.Close
End Sub

PROGRAM SERVER GERBANG TOL KELUAR PASTEUR

Private Sub bntConnect_Click()
On Error GoTo t
sock1.Close
sock1.RemoteHost = txtIP
sock1.RemotePort = txtPort
sock1.Connect
Exit Sub
t:
MsgBox "Error : " & Err.Description, vbCritical
End Sub
Private Sub bntExit_Click()
End
End Sub
Private Sub bntSend_Click()
On Error GoTo t
sock1.SendData txtSend
txtLog = txtLog & "Client : " & txtSend & vbCrLf
txtSend = ""
Exit Sub
t:
Lampiran

MsgBox "Error : " & Err.Description
sock1_Close
End Sub
Private Sub DEC_Change()
DECHEX.Text = Hex(DEC.Text)
End Sub

Private Sub Dec_val_Click()
    txtLog.Text = ""
    decval.Text = ""
    Timer5.Enabled = True
    On Error GoTo t
    Do While (Len(DECHEX.Text) < 8)
        DECHEX.Text = "0" & DECHEX.Text
    Loop
    sock1.SendData "-" & Block2.Text & DECHEX.Text
    Exit Sub
    t:
    MsgBox "Error : " & Err.Description
    sock1_Close
    End Sub

Private Sub Form_Load()
    Picture2.Left = 14100
    ClientFrm.Height = 1335
    ClientFrm.Width = 4800
    End Sub
Private Sub ID_capture_Click()
Lampiran

txtLog.Text = ""
Result_2.Text = ""
Timer2.Enabled = True
On Error GoTo t
sock1.SendData "s"
Exit Sub
t:
MsgBox "Error : " & Err.Description
sock1.Close
End Sub

Private Sub Login_Click()
Result_3.Text = ""
txtLog.Text = ""
Timer3.Enabled = True
On Error GoTo t
sock1.SendData "l" & block.Text & "AA" & KEY
Exit Sub
t:
MsgBox "Error : " & Err.Description
sock1.Close
End Sub

Private Sub Read_Click()
Timer15.Enabled = True


txtLog.Text = ""
RD.Text = ""
Timer6.Enabled = True
On Error GoTo t
sock1.SendData "r" & Block3.Text
Exit Sub

Private Sub Read_val_Click()

txtLog.Text = ""
RV.Text = ""
Timer4.Enabled = True
On Error GoTo t
sock1.SendData "rv" & Block2.Text
Exit Sub

Private Sub RESET_Click()

txtLog.Text = ""
Result_1.Text = ""
Timer1.Enabled = True
On Error GoTo t
sock1.SendData "x"
Exit Sub

Private Sub sock1_Close()
sock1.Close

txtLog = txtLog & "*** Disconnected" & vbCrLf

Timer8.Enabled = True

ClientFrm.Height = 1335

ClientFrm.Width = 4800

End Sub

Private Sub sock1_Connect()

    txtLog = "Connected to " & sock1.RemoteHostIP & vbCrLf

    Timer8.Enabled = False

    ClientFrm.Height = 6045

    ClientFrm.Width = 15000

End Sub

Private Sub sock1_DataArrival(ByVal bytesTotal As Long)

    Dim dat As String

    sock1.GetData dat, vbString

    If Timer1.Enabled = True Then Result_1.Text = Result_1.Text + dat

    If Timer2.Enabled = True Then Result_2.Text = Result_2.Text + dat

    If Timer3.Enabled = True Then Result_3.Text = Result_3.Text + dat

    If Timer4.Enabled = True Then RV.Text = RV.Text + dat

    If Timer5.Enabled = True Then decval.Text = decval.Text + dat

    If Timer6.Enabled = True Then RD.Text = RD.Text + dat

    If Timer7.Enabled = True Then WRT.Text = WRT.Text + dat

    txtLog = txtLog & dat & vbCrLf

End Sub

Private Sub sock1_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean)

    txtLog = txtLog & "*** Error : " & Description & vbCrLf

End Sub

Universitas Kristen Maranatha
Private Sub Timer1_Timer()
    Timer1.Enabled = False
End Sub

Private Sub Timer10_Timer()
    If Timer8.Enabled = False Then
        Call RESET_Click
    End If
    Timer10.Enabled = False
End Sub

Private Sub Timer11_Timer()
    If Timer8.Enabled = False Then
        Call ID_capture_Click
    End If
    Timer11.Enabled = False
End Sub

Private Sub Timer12_Timer()
    If Timer8.Enabled = False Then
        Call Login_Click
    End If
    Timer12.Enabled = False
End Sub

Private Sub Timer13_Timer()
If Timer8.Enabled = False Then
Call Read_val_Click
Timer13.Enabled = False
End If
End Sub

Private Sub Timer14_Timer()
If Timer8.Enabled = False Then
Call Read_Click
Timer14.Enabled = False
End If
End Sub

Private Sub Timer15_Timer()
If Left(RD.Text, 32) = "000000000000504144414C4152414E47" Then
DEC.Text = "2000"
Call Dec_val_Click
End If
End If

If Left(RD.Text, 32) = "0000000000000000000043494D414849" Then
DEC.Text = "1000"
Call Dec_val_Click
End If
asal.Caption = "CIMAHI"
Picture2.Left = -120
End If

Timer15.Enabled = False
End Sub

Private Sub Timer16_Timer()
If Timer8.Enabled = False Then
Call Write_Click
Timer16.Enabled = False
End If
End Sub

Private Sub Timer2_Timer()
Timer2.Enabled = False
End Sub

Private Sub Timer3_Timer()
Timer3.Enabled = False
End Sub

Private Sub Timer4_Timer()
Timer4.Enabled = False
End Sub
Private Sub Timer5_Timer()
Timer5.Enabled = False
End Sub
Private Sub Timer6_Timer()
Timer6.Enabled = False
End Sub
Private Sub Timer7_Timer()
Timer7.Enabled = False
End Sub
Private Sub Timer8_Timer()
Call bntConnect_Click
End Sub
Private Sub Timer9_Timer()
Text3.Text = (Val(Mid(RV.Text, 1, 1)) * 268435456) + (Val(Mid(RV.Text, 2, 1)) * 16777216) + (Val(Mid(RV.Text, 3, 1)) * 1048576) + (Val(Mid(RV.Text, 4, 1)) * 65536) + (Val(Mid(RV.Text, 5, 1)) * 4096) + (Val(Mid(RV.Text, 6, 1)) * 256) + (Val(Mid(RV.Text, 7, 1)) * 16) + (Val(Mid(RV.Text, 8, 1)))
For i = 1 To 8
Text2.Text = 0
Select Case Mid(RV.Text, i, 1)
Case "A"
Text2.Text = 10
Case "B"
Text2.Text = 11
Case "C"
Text2.Text = 12
Case "D"
Text2.Text = 13
Case "E"
Text2.Text = 14
Case "F"
Text2.Text = 15
End Select
Select Case i
Case 1
Case 2
Case 3
Case 4
Case 5
Case 6
Case 7
Case 8
End Select
Next i

Text4.Text = (Val(Mid(decval.Text, 1, 1)) * 268435456) + (Val(Mid(decval.Text, 2, 1)) * 16777216) + (Val(Mid(decval.Text, 3, 1)) * 1048576) + (Val(Mid(decval.Text, 4, 1)) * 65536) + (Val(Mid(decval.Text, 5, 1)) * 4096) + (Val(Mid(decval.Text, 6, 1)) * 256) + (Val(Mid(decval.Text, 7, 1)) * 16) + (Val(Mid(decval.Text, 8, 1)))
For $i = 1$ To 8
Text1.Text = 0
Select Case Mid(decval.Text, $i$, 1)
    Case "A"
        Text1.Text = 10
    Case "B"
        Text1.Text = 11
    Case "C"
        Text1.Text = 12
    Case "D"
        Text1.Text = 13
    Case "E"
        Text1.Text = 14
    Case "F"
        Text1.Text = 15
End Select
Select Case $i$
    Case 1
    Case 2
    Case 3
    Case 4
    Case 5
    Case 6
Case 7

Case 8
End Select
Next i

End Sub

Private Sub Write_Click()

wrtval.Text = ""

i = Len(Text5.Text)
For y = 1 To i
wrtval.Text = Hex(Asc(Right(Text5.Text, y))) + wrtval.Text
Next y

Do While (Len(wrtval.Text) < 32)
wrtval.Text = "0" & wrtval.Text
Loop

txtLog.Text = ""
WRT.Text = ""
Timer7.Enabled = True
On Error GoTo t
sock1.SendData "w" & Block3.Text & wrtval.Text
Exit Sub

Universitas Kristen Maranatha
MsgBox "Error : " & Err.Description
sock1_Close
End Sub
LAMPIRAN B

ACR120S SDK User Manual
CONTENTS

Scopes ................................................................................................................................. 3
USB Interface ....................................................................................................................... 3
Group A. Reader Commands ................................................................................................. 4
  1. ACR120_Open .................................................................................................................. 4
  2. ACR120_Close ................................................................................................................ 4
  3. ACR120_Reset ................................................................................................................ 4
  4. ACR120_Status .............................................................................................................. 6
  5. ACR120_ReadRC531Reg ............................................................................................... 8
  6. ACR120_WriteRC531Reg ............................................................................................. 8
  7. ACR120_DirectSend .................................................................................................... 10
  8. ACR120_DirectReceive .............................................................................................. 10
  9. ACR120_RequestDLLVersion ...................................................................................... 11
 10. ACR120_ReadEEPROM ............................................................................................. 12
 11. ACR120_WriteEEPROM ............................................................................................. 12
 12. ACR120_ReadUserPort ............................................................................................... 14
 13. ACR120_WriteUserPort .............................................................................................. 14
 14. ACR120_Power ........................................................................................................... 16
Group B. General Card Commands .................................................................................... 17
  1. ACR120_Select ............................................................................................................. 17
  2. ACR120_ListTags ........................................................................................................ 18
  3. ACR120_MULTI_TAG_SELECT .................................................................................... 19
  4. ACR120_TxDataTelegram .......................................................................................... 21
  1. ACR120_Login ............................................................................................................ 23
  2. ACR120_Read ................................................................................................................ 25
  3. ACR120_ReadValue .................................................................................................... 25
  4. ACR120_Write ............................................................................................................. 26
  5. ACR120_WriteValue .................................................................................................. 26
  6. ACR120_WriteMasterKey ......................................................................................... 28
  7. ACR120_Inc .................................................................................................................. 28
  8. ACR120_Dec ................................................................................................................ 29
  9. ACR120_Copy ............................................................................................................ 29
Group D. Card Commands for ASK CTS256B/512B Cards (Only for some special versions) .................................................................................................................. 31
  1. ACR120_ASKSectorWrite .......................................................................................... 31
  2. ACR120_ASKSectorRead ......................................................................................... 31
  3. ACR120_ASKSectorMultiRead (for CTS512B only) .................................................. 32
Appendix: ............................................................................................................................ 33
  1. Error Codes returned by High Level APIs ................................................................. 33
  2. Possible TAG Types .................................................................................................... 35
  3. USB ID and Drivers for ACR120U ............................................................................ 35
  5. Physical and Logical Block/Sector Calculation ....................................................... 37
     1. Mifare 1K .............................................................................................................. 37
     2. Mifare 4K .............................................................................................................. 37
SCOPES

The ACR120U USB High Level APIs are some standard functions for controlling the Reader and accessing the supported contactless-cards. By using the High Level APIs, the users can develop applications that involve the use of contactless-cards with minimum effort. For examples,

- Access control, Identification: Reading the serial numbers of all cards in the field.
- Data Storage: Performing encrypted read and write operations.
- Ticketing: Performing read, write, increment and decrement operations in an encrypted environment.
- Multi applications: Performing read, write, increment and decrement operations on various sectors of the card.

# The High Level APIs are available for Windows 98, ME, 2000 & XP Operating Systems.

USB INTERFACE

The ACR120U is connected to a computer through USB as specified in the USB Specification 1.1. The ACR120U is working in low speed mode, i.e. 1.5 Mbps.

USB Interface Wiring

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( V_{\text{BUS}} )</td>
<td>+5V power supply for the reader (~100mA)</td>
</tr>
<tr>
<td>2</td>
<td>( \text{D-} )</td>
<td>Differential signal transmits data between ACR120U and PC.</td>
</tr>
<tr>
<td>3</td>
<td>( \text{D+} )</td>
<td>Differential signal transmits data between ACR120U and PC.</td>
</tr>
<tr>
<td>4</td>
<td>( \text{GND} )</td>
<td>Reference voltage level for power supply</td>
</tr>
</tbody>
</table>

NOTE - In order for the ACR120U functioning properly through USB interface, ACS proprietary device drive has to be installed. Please refer to the Device Driver Installation Guide for more detail.
GROUP A. READER COMMANDS

1. ACR120_Open

High Level API:

`DLLAPI INT16 AC_DECL ACR120_Open(INT16 ReaderPort);`

<table>
<thead>
<tr>
<th>Description</th>
<th>To open a port (connection) to Reader.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters ReaderPort</td>
<td>The port number. Available choices are &quot;ACR120_USB1&quot; to &quot;ACR120_USB8&quot;.</td>
</tr>
<tr>
<td>Return Value INT16</td>
<td>Handle for further operations. Error Code &lt; 0</td>
</tr>
</tbody>
</table>

2. ACR120_Close

High Level API:

`DLLAPI INT16 AC_DECL ACR120_Close(INT16 hReader);`

<table>
<thead>
<tr>
<th>Description</th>
<th>To close the port (connection) to Reader.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>Return Value INT16</td>
<td>0 = success; Error Code &lt; 0</td>
</tr>
</tbody>
</table>

3. ACR120_Reset

High Level API:

`DLLAPI INT16 AC_DECL ACR120_Reset(INT16 hReader);`

<table>
<thead>
<tr>
<th>Description</th>
<th>To reset the Mifare Chip of the Reader, then restore the factory settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>Return Value INT16</td>
<td>0 = success; Error Code &lt; 0</td>
</tr>
</tbody>
</table>

© Advanced Card Systems Ltd.
Sample Code:

```c
#include "acr120.h"

main()
{
    // Open a communication channel, the first USB Reader
    INT16 hReader=ACR120_Open(ACR120_USB1);

    // Reset the Reader to the initial state.
    if(hReader>0)
    {
        INT16 Status= ACR120_Reset(hReader);
    }
    else
    {
        // error happened
    }

    // some operations
    // Close the communication channel, the first USB Reader
    if(hReader>0)
    {
        Status= ACR120_Close(hReader);
        hReader = -1;
    }
}
```
GROUP A. READER COMMANDS

4. ACR120_Status

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_Status(INT16     hReader,
               UINT8    pFirmwareVersion[20],
               STRUCT_STATUS pReaderStatus);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Return the firmware version and the Reader status.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>hReader: The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td></td>
<td>pFirmwareVersion: The firmware version will be returned (20 bytes)</td>
</tr>
<tr>
<td></td>
<td>pReaderStatus: The Reader status.</td>
</tr>
<tr>
<td>Return Value</td>
<td>INT16: 0 = success; Error Code &lt; 0</td>
</tr>
</tbody>
</table>

Sample Code:

```c
#include "acr120.h"

// Obtain the Firmware version & Reader Status if the USB connection is already established
if(hReader>0)
{
    UINT8 FirmwareVersion[20];
    STRUCT_STATUS ReaderStatus;

    INT16 Status= ACR120_Status(hReader, FirmwareVersion, &ReaderStatus);

    If(Status== SUCCESS_READER_OP)
    {
        // do some operations if the operation is success
    }
    else
    {
        // error happened!!
    }
}
```
GROUP A. READER COMMANDS

(Cont.)

Struct STRUCT_STATUS
{
    // 0x01 = Type A; 0x02 = Type B; 0x03 = Type A + Type B
    UINT8 MifareInterfaceType;

    // Bit 0 = Mifare Light; Bit 1 = Mifare1K; Bit 2 = Mifare 4K; Bit 3 = Mifare DESFire
    // Bit 4 = Mifare UltraLight; Bit 5 = JCOP30; Bit 6 = Shanghai Transport
    // Bit 7 = MPCOS Combi; Bit 8 = ISO type B, Calypso
    // Bit 9 – Bit 31 = To be defined
    UINT32 CardsSupported;

    UINT8 CardOpMode; // To be defined

    UINT8 FWI; // the current FWI value (time out value)

    UINT8 RFU; // To be defined

    UINT16 RFU2; // to be defined

} ReaderStatus;
GROUP A. READER COMMANDS

5. ACR120_ReadRC531Reg

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_ReadRC531Reg(INT16 hReader,  
                      UINT8 RegNo,  
                      UINT8* pValue);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>To read the Mifare registers.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>RegNo</td>
<td>The register number.</td>
</tr>
<tr>
<td>pValue</td>
<td>Mifare register’s value.</td>
</tr>
<tr>
<td>Return Value</td>
<td>INT16</td>
</tr>
<tr>
<td></td>
<td>Result code. 0 means success.</td>
</tr>
</tbody>
</table>

6. ACR120_WriteRC531Reg

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_WriteRC531Reg(INT16 hReader,  
                      UINT8 RegNo,  
                      UINT8 Value);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>To write the Mifare registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>RegNo</td>
<td>The register number.</td>
</tr>
<tr>
<td>Value</td>
<td>Mifare register’s value to write</td>
</tr>
<tr>
<td>Return Value</td>
<td>INT16</td>
</tr>
<tr>
<td></td>
<td>Result code. 0 means success.</td>
</tr>
</tbody>
</table>
Sample Code:

```c
#include "acr120.h"

// Read & Write the Reader Register if the USB connection is already established
if(hReader>0)
{
    UINT8 RegNo=0x05; // the register address
    UINT8 Value;  // the register value

    INT16 Status= ACR120_ReadRC531Reg(hReader, RegNo, &Value);

    If(Status== SUCCESS_READER_OP)
    {
        // Update the register value
        Value!=0x01;
        Status= ACR120_WriteRC531Reg(hReader, RegNo, Value);
    }

    if(Status!= SUCCESS_READER_OP)
    {
        // error happened!!
    }
}

#Users are not recommended to modify the internal register setting.
```
GROUP A. READER COMMANDS

7. ACR120_DirectSend

High Level API:

DLLAPI INT16 AC_DECL
ACR120_DirectSend(INT16 hReader,
      UINT8 DataLength,
      UINT8* pData,
      UINT8* pResponseDataLength,
      UINT8* pResponseData,
      UINT16 TimedOut);

<table>
<thead>
<tr>
<th>Description</th>
<th>To send data to the Reader directly.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>DataLength (N)</td>
<td>The Data Length (maximum 66 bytes)</td>
</tr>
<tr>
<td>Data</td>
<td>The Data to be sent</td>
</tr>
<tr>
<td>pResponseDataLength (K)</td>
<td>The Response Data Length</td>
</tr>
<tr>
<td>pResponseData</td>
<td>The Response Data</td>
</tr>
<tr>
<td>TimedOut</td>
<td>The Time Out for waiting the response data in m-sec</td>
</tr>
</tbody>
</table>

| Return Value      | INT16 0 = success; Error Code < 0  |

8. ACR120_DirectReceive

High Level API:

DLLAPI INT16 AC_DECL
ACR120_DirectReceive(INT16 hReader,
      UINT8 RespectedDataLength,
      UINT8* pReceivedDataLength,
      UINT8* pReceivedData,
      UINT16 TimedOut);

<table>
<thead>
<tr>
<th>Description</th>
<th>To receive data from the Reader directly.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>RespectedDataLength</td>
<td>The Respected Data Length to be received (maximum 64 bytes)</td>
</tr>
<tr>
<td>pReceivedDataLength (K)</td>
<td>The Data Length of the received data</td>
</tr>
<tr>
<td>pReceivedData</td>
<td>The Received Data</td>
</tr>
<tr>
<td>TimedOut</td>
<td>The Time Out for waiting the received data in m-sec</td>
</tr>
</tbody>
</table>

| Return Value      | INT16 0 = success; Error Code < 0  |

# These two APIs are for special purposes.
GROUP A. READER COMMANDS

9. ACR120_RequestDLLVersion

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_RequestDLLVersion(UINT8* pVersionInfoLength,
                          UINT8* pVersionInfo);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>To get the reader’s API DLL version information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>pVersionInfoLength</td>
<td>It returns the DLL Version string.</td>
</tr>
<tr>
<td>pVersionInfo</td>
<td></td>
</tr>
</tbody>
</table>

| Return Value | INT16       | 0 = success; Error Code < 0 |

Sample Code:

```
#include “acr120.h”

// Get the DLL Version

UINT8 Length;
UINT8 Version[40]; // the DLL Version string is less than 40 bytes long

INT16 Status=ACR120_RequestDLLVersion(&Length, Version);

if(Status== SUCCESS_READER_OP)
{
    // display the DLL version,

    Version[Length]=‘\0’; // add the terminator ‘\0’
    printf(“The DLL version is %s”, Version);
}
else
{
    // DLL Error !!
}
```
GROUP A. READER COMMANDS

10. ACR120_ReadEEPROM

High Level API:

```c
DLLAPI INT16 AC_DECL ACR120_ReadEEPROM(INT16 hReader,
                                        UINT8 RegNo,
                                        UINT8* pEEPROMData);
```

**Description**
Read the internal EEPROM.

**Parameters**
- **hReader**: The handle to the Reader returned by ACR120_Open().
- **RegNo**: The register number.
- **pEEPROMData**: Contain the EEPROM register’s value.

**Return Value**
INT16 Result code. 0 means success.

11. ACR120_WriteEEPROM

High Level API:

```c
DLLAPI INT16 AC_DECL ACR120_WriteEEPROM(INT16 hReader,
                                        UINT8 RegNo,
                                        UINT8 EEPROMData);
```

**Description**
Write the internal EEPROM.

**Parameters**
- **hReader**: The handle to the Reader returned by ACR120_Open().
- **RegNo**: The register number.
- **EEPROMData**: The EEPROM register’s value to write.

**Return Value**
INT16 Result code. 0 means success.
Sample Code:

```c
#include "acr120.h"

// Read & Write the EEPROM if the USB connection is already established
if(hReader>0)
{
    UINT8 Address=0x04; // the address of the EEPROM to be accessed
    UINT8 Value;       // the value

    INT16 Status = ACR120_ReadEEPROM(hReader, Address, &Value);

    if(Status== SUCCESS_READER_OP)
    {
        // Update the register value
        Value &= 0x0F;
        Status = ACR120_WriteEEPROM(hReader, Address, Value);
    }

    if(Status!= SUCCESS_READER_OP)
    {
        // error happened!!
    }
}
```
GROUP A. READER COMMANDS

12. ACR120_ReadUserPort

High Level API:

```c
DLLAPI INT16 AC_DECL
ACR120_ReadUserPort(INT16 hReader,
UINT8* pUserPortState);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Read in the state of user port.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>pUserPortState</td>
<td>Contain the port state (only Bit 2 &amp; Bit 6 are used).</td>
</tr>
<tr>
<td>Return Value</td>
<td>INT16</td>
</tr>
<tr>
<td>Result code.</td>
<td>0 means success.</td>
</tr>
</tbody>
</table>

13. ACR120_WriteUserPort

High Level API:

```c
DLLAPI INT16 AC_DECL
ACR120_WriteUserPort(INT16 hReader,
UINT8 UserPortState);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Update the state of user port.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>UserPortState</td>
<td>Contain the port state to write (only Bit 2 &amp; Bit 6 are used).</td>
</tr>
<tr>
<td>Return Value</td>
<td>INT16</td>
</tr>
<tr>
<td>Result code.</td>
<td>0 means success.</td>
</tr>
</tbody>
</table>

**UserPortState:**

- Bit 0: Not Used
- Bit 1: Not Used
- Bit 2: Buzzer (0 = OFF; 1 = ON)
- Bit 3: Not Used
- Bit 4: Not Used
- Bit 5: Not Used
- Bit 6: LED (0 = OFF; 1 = ON)
- Bit 7: Not Used
Sample Code:

```c
#include "acr120.h"

// Turn on the LED if the USB connection is already established
if(hReader>0)
{
    UINT8 PortValue;  // the value of the user port

    INT16 Status = ACR120_ReadUserPort(hReader, &PortValue);

    If(Status== SUCCESS_READER_OP)
    {
        // Turn on the LED only
        PortValue |= 0x40;
        Status = ACR120_WriteUserPort(hReader, PortValue);
    }
}
```
GROUP A. READER COMMANDS

14. ACR120_Power

High Level API:

```c
DLLAPI INT16 AC_DECL
ACR120_Power(INT16  hReader,
              INT8  State);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Turn on or off the antenna power.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameters</strong></td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>State</td>
<td>Turn OFF (0) or ON (1).</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>INT16</td>
</tr>
<tr>
<td></td>
<td>Result code. 0 means success.</td>
</tr>
</tbody>
</table>

Sample Code:

```c
#include "acr120.h"

// Turn off the Antenna Power for power saving
if(hReader>0)
{
    INT16 Status= ACR120_Power(hReader, 0x00);
}

// The Antenna Power will be turned on automatically if any Card Operations is started. // E.g. ACR120_Select(). Don’t need to turn on the Antenna Power manually. // However, the Antenna Power cannot be turned off while any Card Operations is running.
```
GROUP B. GENERAL CARD COMMANDS

NOTE - All Card API’s involving SECTOR and BLOCK parameters please refer to APPENDIX 5 for further explanation

1. ACR120_Select

High Level API:

DLLAPI INT16 ACR_DECL
ACR120_Select(INT16 hReader,
 UINT8* pResultTagType,
 UINT8* pResultTagLength,
 UINT8 pResultSN[10]);

<table>
<thead>
<tr>
<th>Description</th>
<th>Select a single card and return the card ID (Serial Number)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>pResultTagType</td>
<td>Contain the selected Tag Type</td>
</tr>
<tr>
<td>pResultTagLength</td>
<td>Contain the Length of the selected TAG.</td>
</tr>
<tr>
<td>pResultSN</td>
<td>If the pResultTagLength = 4 or 7 or 10, the pSN contains the selected card ID (Serial Number). The ID may be 4 or 7 or 10 Bytes long.</td>
</tr>
</tbody>
</table>

Return Value | INT16  | Result code. 0 means success.

Sample Code:

```c
#include "acr120.h"

// Select a TAG on the reader
if(hReader>0)
{
    UINT8 TagType;      // the Tag Type
    UINT8 TagLength;    // the length of the Tag SN
    UINT8 TagSN[10];    // The SN of the Tag

    // This API is useful for selecting a TAG in which the SN is not known in advance.
    INT16 Status = ACR120_Select(hReader, &TagType, &TagLength, TagSN);

    if(Status==SUCCESS_READER_OP)
    {
        // Now the TagSN[10] contains the SN of the Tag
        // Please check the TagLength to determine the actual length of the SN
        // e.g for Mifare 1K card, the TagLength will be equal to 0x04.
        // the TagType will be equal to 0x02;
    }
    else
    {
        // No TAG is found!!
    }
}
```

© Advanced Card Systems Ltd.
GROUP B. GENERAL CARD COMMANDS

2. ACR120_ListTags

High Level API:

DLLAPI INT16 AC_DECL

ACR120_ListTags(INT16  hReader,
                UINT8*  pNumTagFound,
                UINT8  pTagType[4],
                UINT8  pTagLength[4],
                UINT8  pSN[4][10]);

Description
List out the serial numbers of all tags, which are in readable antenna range.

Parameters
- **hReader**: The handle to the Reader returned by ACR120_Open().
- **pNumTagFound**: Contains of number of TAG listed.
- **pTagType[4]**: Contains the TAG Type
- **pTagLength[4]**: Contains the length of the serial number.
- **pSN[4][10]**: The flat array of serial numbers. All serial numbers are concatenated with fixed length – 10 bytes.

Return Value
- **INT16**: Result code. 0 means success.

Sample Code:

```c
#include "acr120.h"

UINT8 TagFound;  // number of TAG found
UINT8 TagType[4]; // the Tag Type
UINT8 TagLength[4]; // the length of the Tag SN
UINT8 TagSN[4][10]; // The SN of the Tag

// Find all the TAGs placed on the reader antenna. Maximum 4 TAGs can be recognized
// by the reader at the same time.
INT16 Status= ACR120_ListTags(hReader,
                              &TagFound, TagType, TagLength, TagSN);

if(Status== SUCCESS_READER_OP)
{
    // Now the TagFound contains the number of TAG recognized by the reader

    // Assume the TagFound is equal to two, Two TAGs are found
    // the TagSN[0][10] contains the SN of the first Tag
    // the TagLength[0] contains the actual length of the SN of the first TAG
    // the TagType[0] contains the TAG Type of the first TAG

    // the TagSN[1][10] contains the SN of the second Tag
    // the TagLength[1] contains the actual length of the SN of the second TAG
    // the TagType[1] contains the TAG Type of the second TAG

    // the content of TagSN[2][10], TagLength[2], TagType[2] have no meaning
    // Similarly, the content of TagSN[3][10], TagLength[3], TagType[3] have no
    // meaning

} else { // No TAG is found!! }
```
GROUP B. GENERAL CARD COMMANDS

3. ACR120_MultiTagSelect

High Level API:

`DLLAPI INT16 AC_Decl
ACR120_MultiTagSelect(INT16  hReader,
UINT8  TagLength,
UINT8  SN[10],
UINT8*  pResultTagType,
UINT8*  pResultTagLength,
UINT8*  pResultSN);`

**Description**
To select a TAG with specific serial number.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>TagLength (N)</td>
<td>Contains the length of the serial number of the TAG to be selected. The TagLength may be 4, 7 or 10 bytes long.</td>
</tr>
<tr>
<td>SN</td>
<td>Contain the serial number of the TAG to be selected.</td>
</tr>
<tr>
<td>pResultTagType</td>
<td>Contain the selected Tag Type</td>
</tr>
<tr>
<td>pResultTagLength (K)</td>
<td>Contain the length of the serial number of the selected TAG. The pResultTagLength may be 4, 7 or 10 bytes long.</td>
</tr>
<tr>
<td>pResultSN</td>
<td>The serial number of the selected TAG.</td>
</tr>
</tbody>
</table>

**Return Value**
INT16 Result code. 0 means success.
Sample Code:

```c
#include "acr120.h"

UINT8 ResultTagType;  // the Tag Type detected by the reader
UINT8 ResultTagLength;  // the Tag length detected by the reader
UINT8 ResultTagSN[10];  // the Tag SN detected by the reader

// The SN of the Tag is “A6 2D EA 92”, the length is 4 bytes
// Fill the rest of the array with zeros
UINT8 TagSN[10]={ 0xA6, 0x2D, 0xEA, 0x92, 0x00,
                  0x00, 0x00, 0x00, 0x00, 0x00};

// Select an arbitrary TAG if the SN of the TAG is known already. E.g. By using
// ACR120_ListTags()
// This API is useful for selecting an arbitrary TAG among all the TAGs.

INT16 Status= ACR120_MultiTagSelect(hReader,
                                      0x04, TagSN,
                                      ResultTagType, ResultTagLength, ResultTagSN);

If(Status== SUCCESS_READER_OP)
{
    // the ResultTagSN[10] contains the SN of the Tag detected by the reader
    // it must be the same as the TagSN[10]

    // the ResultTagLength contains the actual length of the SN of the TAG detected by
    // the reader. it must be the same as the TagLength

    // the ResultTagType contains the TAG Type of the TAG detected by the reader
}
else
{
    // No TAG is selected!!
}
GROUP B. GENERAL CARD COMMANDS

4. ACR120_TxDataTelegram

High Level API:

```c
DLLAPI INT16 AC_DECL
ACR120_TxDataTelegram(INT16  hReader,
                        UINT8  SendDataLength,
                        UINT8* pSendData
                        UINT8* pReceivedDataLength,
                        UINT8* pReceivedData);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Send data to the Selected Card.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>SendDataLength (N)</td>
<td>The length of the data to be sent</td>
</tr>
<tr>
<td>pSendData</td>
<td>The data to be sent</td>
</tr>
<tr>
<td>pReceivedDataLength (K)</td>
<td>The length of the received data</td>
</tr>
<tr>
<td>pReceivedData</td>
<td>The received data</td>
</tr>
<tr>
<td>Return Value</td>
<td>INT16 Result code. 0 means success.</td>
</tr>
</tbody>
</table>

Sample Code: None (please refer to the related document for more detailed information)

The Parameter “SendData” has the following format:

<table>
<thead>
<tr>
<th>Telegram Length (1 Byte)</th>
<th>Option Byte (1 Byte)</th>
<th>Data (K Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>#</td>
<td>Telegram Data</td>
</tr>
</tbody>
</table>

**Telegram Length (K):** This byte is transferred too for compatibility reasons even though it could be calculated with the SendDataLength. **SendDataLength (N) = Telegram Length (K) + 2**

**Option byte:**
This byte holds transfer options.
- Bit 0: if set Parity generation is enabled
- Bit 1: if set Parity is odd, otherwise Parity bit is even
- Bit 2: if set CRC generation for transmission is enabled
- Bit 3: if set CRC checking for receiving is enabled
- Bit 4: if set Crypto unit is deactivated before transmission start
  - Activation of the Crypto unit is only possible by using the login instruction
- Bit 5,6,7: Bit Framing (Number of Bits from last Byte transmitted)

**Data:** The telegram data to be sent
Sample Code:

E.g. To send "RATS". {0x02, 0x0F, 0xE0, 0x50}

In which,
0x02: The DataTelegram Length
0x0F: The DataTelegram Option. Pls refer to the API Document for more detailed info.
{0xE0, 0x50}: RATS Command <DataTelegram to be sent>

// Sample Code for sending "RATS" to DESFire Card

```c
UINT8 GetRATS[]={0x02, 0x0F, 0xE0, 0x50};
UINT8 BlockData[64], BlockDataLength;

CMDStatus=ACR120_TxDataTelegram(ReaderHandle, 0x04, GetRATS, &BlockDataLength, BlockData);

// If the command is successfully executed,
// the BlockDataLength will be equal to 0x06
// And the Block Data will have the data {0x06, 075, 0x77, 0x81, 0x02, 0x80}
```

#Common TeleDatagram Option Bytes Setting

- MIFare 1K/4K: 0xF3
- DESFire: 0x0F
- ISO Type B: 0x0C
1. ACR120_Login

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_Login(INT16  hReader,
              UINT8  Sector,
              UINT8  KeyType,
              INT8  StoredNo,
              UINT8  pKey[6]);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Perform an authentication to access one sector of the card. Only one sector can be accessed at a time.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>hReader The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td></td>
<td>Sector The sector no. to login.</td>
</tr>
<tr>
<td></td>
<td>KeyType The type of key. It can be</td>
</tr>
<tr>
<td></td>
<td>AC_MIFARE_LOGIN_KEYTYPE_A,</td>
</tr>
<tr>
<td></td>
<td>AC_MIFARE_LOGIN_KEYTYPE_B,</td>
</tr>
<tr>
<td></td>
<td>AC_MIFARE_LOGIN_KEYTYPE_DEFAULT_A,</td>
</tr>
<tr>
<td></td>
<td>AC_MIFARE_LOGIN_KEYTYPE_DEFAULT_B,</td>
</tr>
<tr>
<td></td>
<td>AC_MIFARE_LOGIN_KEYTYPE_DEFAULT_F,</td>
</tr>
<tr>
<td></td>
<td>AC_MIFARE_LOGIN_KEYTYPE_STORED_A and</td>
</tr>
<tr>
<td></td>
<td>AC_MIFARE_LOGIN_KEYTYPE_STORED_B</td>
</tr>
<tr>
<td>StoredNo</td>
<td>The stored no of key if KeyType =</td>
</tr>
<tr>
<td></td>
<td>AC_MIFARE_LOGIN_KEYTYPE_STORED_A or AC_MIFARE_LOGIN_KEYTYPE_STORED_B</td>
</tr>
<tr>
<td>pKey</td>
<td>The login key if KeyType =</td>
</tr>
<tr>
<td></td>
<td>AC_MIFARE_LOGIN_KEYTYPE_A or AC_MIFARE_LOGIN_KEYTYPE_B. It’s AC_MIFARE_KEY_LEN(6) bytes long.</td>
</tr>
<tr>
<td>Return Value</td>
<td>INT16 Result code. 0 means success.</td>
</tr>
</tbody>
</table>

Constant Definition:

- `AC_MIFARE_LOGIN_KEYTYPE_A` 0xAA
- `AC_MIFARE_LOGIN_KEYTYPE_B` 0xBB
- `AC_MIFARE_LOGIN_KEYTYPE_DEFAULT_A` 0xAD
- `AC_MIFARE_LOGIN_KEYTYPE_DEFAULT_B` 0xBD
- `AC_MIFARE_LOGIN_KEYTYPE_DEFAULT_F` 0xFD
- `AC_MIFARE_LOGIN_KEYTYPE_STORED_A` 0xAF
- `AC_MIFARE_LOGIN_KEYTYPE_STORED_B` 0xBF
Sample Code:

```
#include "acr120.h"

// Login the selected TAG on the reader
if(hReader>0)
{
    UINT8 TagType;  // the Tag Type
    UINT8 TagLength;  // the length of the Tag SN
    UINT8 TagSN[10];  // The SN of the Tag

    // Select a Tag
    INT16 Status= ACR120_Select(hReader, &TagType, &TagLength, TagSN);

    // Assume a Tag is successfully selected
    // Login the Sector 0x02 with a given key (Key A Login)
    UINT8 Key[6]={ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06}; // the key used for login
    Status= ACR120_Login(hReader, 0x02,
        AC_MIFARE_LOGIN_KEYTYPE_A, 0, Key);
    If(Status== SUCCESS_READER_OP)
    {
        // Now the Sector 0x02 is successfully authenticated (login success)
    }
    else
    {
        // The Sector 0x02 is not authenticated (login fail)!!
    }

    // some operations
    //
    //

    // Assume the Tag is still selected
    // Login the Sector 0x08 with a MasterKey 0x01 stored in Reader (Key B Login)
    Status= ACR120_Login(hReader, 0x08,
        AC_MIFARE_LOGIN_KEYTYPE_STORED_B, 0x01, NULL);
    If(Status== SUCCESS_READER_OP)
    {
        // Now the Sector 0x08 is successfully authenticated (login success)
    }
    else
    {
        // The Sector 0x08 is not authenticated (login fail)!!
    }
}
```
GROUP C. CARD COMMANDS FOR MIFARE 1K/4K CARDS

2. ACR120_Read

High Level API:

\[
\text{DLLAPI INT16 AC\_DECL}
\]
\[
\text{ACR120\_Read(IN}T16 \quad \text{hReader,}
\]
\[
\text{UINT8 Block,}
\]
\[
\text{UINT8 pBlockData[16]};
\]

<table>
<thead>
<tr>
<th>Description</th>
<th>Read a block.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>Block</td>
<td>The block number.</td>
</tr>
<tr>
<td>pBlockData</td>
<td>Contain the data read. It’s AC_MIFARE_DATA_LEN(16) bytes long.</td>
</tr>
<tr>
<td>Return Value</td>
<td>INT16 Result code. 0 means success.</td>
</tr>
</tbody>
</table>

3. ACR120_ReadValue

High Level API:

\[
\text{DLLAPI INT16 AC\_DECL}
\]
\[
\text{ACR120\_ReadValue(IN}T16 \quad \text{hReader,}
\]
\[
\text{UINT8 Block,}
\]
\[
\text{INT32* pValueData};
\]

<table>
<thead>
<tr>
<th>Description</th>
<th>Read a value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>Block</td>
<td>The block number.</td>
</tr>
<tr>
<td>pValueData</td>
<td>Contains the value read. It’s 32 bit signed integer.</td>
</tr>
<tr>
<td>Return Value</td>
<td>INT16 Result code. 0 means success.</td>
</tr>
</tbody>
</table>
GROUP C. CARD COMMANDS FOR MIFARE 1K/4K CARDS

4. ACR120_Write

High Level API:

\[
\text{DLLAPI INT16 AC_DECL ACR120_Write(INT16 hReader, UINT8 Block, UINT8 pBlockData[16]);}
\]

| Description | Write a block. |
| Parameters  | hReader  | The handle to the Reader returned by ACR120_Open(). |
| Block       | The block number. |
| pBlockData  | Contain the data to write. It's AC_MIFARE_DATA_LEN(16) bytes long. |

| Return Value | INT16 | Result code. 0 means success. |

5. ACR120_WriteValue

High Level API:

\[
\text{DLLAPI INT16 AC_DECL ACR120_WriteValue(INT16 hReader, UINT8 Block, INT32 ValueData);}
\]

| Description | Write a value. |
| Parameters  | hReader  | The handle to the Reader returned by ACR120_Open(). |
| Block       | The block number. |
| ValueData   | Contain the value to write. It's 32 bit signed integer. |

| Return Value | INT16 | Result code. 0 means success. |
Sample Code:

```c
#include "acr120.h"

// Read & Write the Block if the USB connection is already established
if(hReader>0)
{
    UINT8 BlockData[16] // the data stored in the “Data Block”
    UINT8 BlockValue;  // the value stored in the “Value Block”

    // Assume the sector 0x02 is authenticated already
    // Read the block 0x08 of sector 0x02, each sector contains 4 blocks
    // Sector 0x02 consists of Blocks 0x08, 0x09, 0x0A & 0x0B

    // Assume the Block 0x08 is a “Data Block", read the content
    INT16 Status = ACR120_Read(hReader, 0x08, BlockData);

    // update the block with a new content
    UINT8 NewBlockData[16];
    memset(NewBlockData, 0x00, 16);
    Status = ACR120_Write(hReader, 0x08, NewBlockData);

    // Assume the Block 0x09 is a “Value Block", read the value first
    Status = ACR120_ReadValue(hReader, 0x09, &BlockValue);

    // update the block with a new value. Decrease the value by 50
    Status = ACR120_WriteValue(hReader, 0x09, BlockValue-50);
}
```
GROUP C. CARD COMMANDS FOR MIFARE 1K/4K CARDS

6. ACR120_WriteMasterKey

High Level API:

```c
#include "acr120.h"

// Store a master key into the reader
// There are totally 32 Masterkey storage space in the reader. From location 0x00 to 0x1F
if(hReader>0)
{
    UINT8 MasterKey[6]={0x00, 0x01, 0x02, 0x03, 0x04, 0x05};
    KeyStored=0x01;  // The MasterKey location in the reader.
    Status= ACR120_WriteMasterKey(hReader, KeyStored, MasterKey);
    If(Status== SUCCESS_READER_OP)
    {
        // Now the Masterkey is successfully stored at location 0x01
    }
    else
    {
        // The Masterkey is not stored!!
    }
}
```

### Description
Write master keys.

### Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>KeyNo</td>
<td>The master key number.</td>
</tr>
<tr>
<td>pKey</td>
<td>The key to write. It's AC_MIFARE_KEY_LEN(6) bytes long.</td>
</tr>
</tbody>
</table>

### Return Value
INT16 Result code. 0 means success.
GROUP C. CARD COMMANDS FOR MIFARE 1K/4K CARDS

7. ACR120_Inc

High Level API:

\[
\text{DLLAPI INT16 AC\_DECL} \\
\text{ACR120\_Inc (INT16} \quad h\text{Reader,} \\
\text{UINT8} \quad \text{Block,} \\
\text{INT32} \quad \text{Value,} \\
\text{INT32}\ast \quad p\text{NewValue);}
\]

<table>
<thead>
<tr>
<th>Description</th>
<th>Increment a value block by adding a value.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameters</strong></td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by</td>
</tr>
<tr>
<td>Block</td>
<td>ACR120_Open().</td>
</tr>
<tr>
<td>Value</td>
<td>The block number.</td>
</tr>
<tr>
<td>pNewValue</td>
<td>The updated value after increment.</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>INT16</td>
</tr>
<tr>
<td>Result code.</td>
<td>0 means success.</td>
</tr>
</tbody>
</table>

8. ACR120_Dec

High Level API:

\[
\text{DLLAPI INT16 AC\_DECL} \\
\text{ACR120\_Dec (INT16} \quad h\text{Reader,} \\
\text{UINT8} \quad \text{Block,} \\
\text{INT32} \quad \text{Value,} \\
\text{INT32}\ast \quad p\text{NewValue);}
\]

<table>
<thead>
<tr>
<th>Description</th>
<th>Decrement a value block by subtracting a value.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parameters</strong></td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>Block</td>
<td>The block number.</td>
</tr>
<tr>
<td>Value</td>
<td>The value subtracts.</td>
</tr>
<tr>
<td>pNewValue</td>
<td>The updated value after decrement.</td>
</tr>
<tr>
<td><strong>Return Value</strong></td>
<td>INT16</td>
</tr>
<tr>
<td>Result code.</td>
<td>0 means success.</td>
</tr>
</tbody>
</table>
GROUP C. CARD COMMANDS FOR MIFARE 1K/4K CARDS

9. ACR120_Copy

High Level API:

DLLAPI INT16 AC_DECL
ACR120_Copy(INT16  hReader,
            UINT8  srcBlock,
            UINT8  desBlock,
            INT32*  pNewValue);

Description
Copy a value block to another value block of the same sector.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>srcBlock</td>
<td>The source block number.</td>
</tr>
<tr>
<td>tgtBlock</td>
<td>The target block number.</td>
</tr>
<tr>
<td>pNewValue</td>
<td>The updated value of the desBlock after copy.</td>
</tr>
</tbody>
</table>

Return Value
INT16 Result code. 0 means success.

Sample Code:

```c
#include "acr120.h"

// Read & Write the Value Blocks if the USB connection is already established
if(hReader>0)
{
    UINT8 Block;  // the block number within the sector
    UINT8 BlockValue; // the value stored in the “Value Block”

    // Assume the sector 0x02 is authenticated already
    // each sector contains 4 blocks
    // Sector 0x02 consists of Blocks 0x08, 0x09, 0x0A & 0x0B

    // Assume the Blocks 0x08 and 0x0A are “Value Block”, copy the value from block 0x08 to block 0x0A first.
    INT16 Status= ACR120_Copy(hReader, 0x08, 0x09, &BlockValue);
    // now the BlockValue contains the updated value of Block 0x09

    // update the block 0x0A with a new value. Decrease the value by 100 (decimal)
    Status= ACR120_Dec(hReader, 0x0A, 100, &BlockValue);
    // now the BlockValue contains the updated value of Block 0x0A

    // update the block 0x08 with a new value. Increase the value by 56 (decimal)
    Status= ACR120_Inc(hReader, 0x08, 56, &BlockValue);
    // now the BlockValue contains the updated value of Block 0x08
}
```
GROUP D. CARD COMMANDS FOR ASK CTS256B/512B CARDS (ONLY FOR SOME SPECIAL VERSIONS)

1. ACR120_ASKSectorWrite

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_ASKSectorWrite(INT16  hReader,
                        UINT8  Sector,
                        UINT8  pBlockData[2],
                        UINT8  UpdateMode);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Write a block.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>Sector</td>
<td>The Sector number.</td>
</tr>
<tr>
<td></td>
<td>For CTS256B, 0 &lt;= Sector &lt;= 15</td>
</tr>
<tr>
<td></td>
<td>For CTS512B, 0 &lt;= Sector &lt;= 31</td>
</tr>
<tr>
<td>pBlockData</td>
<td>Contain the data to write. It’s 2 bytes long.</td>
</tr>
<tr>
<td>UpdateMode</td>
<td>‘0’ = Write. It will write ‘1’s at the specified memory location, but not ‘0’. ‘1’ = Update/Erase. It will update the specified location with the data provided. It is the only way to write ‘0’s to the specified memory location</td>
</tr>
</tbody>
</table>

Return Value INT16 Result code. 0 means success.

2. ACR120_ASKSectorRead

High Level API:

```
DLLAPI INT16 AC_DECL
ACR120_ASKSectorRead(INT16  hReader,
                      UINT8  Sector,
                      UINT8  pBlockData[2]);
```

<table>
<thead>
<tr>
<th>Description</th>
<th>Read a block.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>hReader</td>
<td>The handle to the Reader returned by ACR120_Open().</td>
</tr>
<tr>
<td>Sector</td>
<td>The Sector number.</td>
</tr>
<tr>
<td></td>
<td>For CTS256B, 0 &lt;= Sector &lt;= 15</td>
</tr>
<tr>
<td></td>
<td>For CTS512B, 0 &lt;= Sector &lt;= 31</td>
</tr>
<tr>
<td>pBlockData</td>
<td>Contain the data received. It’s 2 bytes long.</td>
</tr>
</tbody>
</table>

Return Value INT16 Result code. 0 means success.
GROUP D. CARD COMMANDS FOR ASK CTS256B/512B CARDS (ONLY FOR SOME SPECIAL VERSIONS)

3. ACR120_ASKSectorMultiRead (for CTS512B only)

High Level API:

\[
\text{DLLAPI INT16 AC_DECL} \\
\text{ACR120_ASKSectorRead(IN16 hReader,} \\
\text{UINT8 Sector,} \\
\text{UINT8 pBlockData[8]);}
\]

<table>
<thead>
<tr>
<th>Description</th>
<th>Read 4 consecutive sector blocks.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>hReader</td>
</tr>
<tr>
<td></td>
<td>Sector</td>
</tr>
<tr>
<td></td>
<td>pBlockData</td>
</tr>
<tr>
<td>Return Value</td>
<td>INT16</td>
</tr>
</tbody>
</table>

Sample Code:

```c
#include "acr120.h"

// Read & Write the Block if the USB connection is already established
if(hReader>0)
{
    UINT8 BlockData[2] ={0x12, 0x34}; // the data stored in the “Data Block”
    UINT8 MultiBlockData[8];

    // Assume a Tag is selected
    // Read the 4 consecutive sector blocks starting from Sector 0x00
    INT16 Status= ACR120_ASKSectorMultiRead(hReader, 0x00, MultiBlockData);

    // Read the content of Sector 0x05
    Status= ACR120_ASKSectorRead(hReader, 0x05, BlockData);
    BlockData[0] |= 0xAA;
    BlockData[1] |= 0x55;

    // Write the new BlockData to Sector 0x05, Write Mode
    Status= ACR120_ASKSectorWrite(hReader, 0x05, BlockData, 0);
    BlockData[0]=0x00; BlockData[1]=0x00;

    // Erase the content of Sector 0x05, Update Mode
    Status= ACR120_ASKSectorWrite(hReader, 0x05, BlockData, 1);
}
```
APPENDIX:

1. Error Codes returned by High Level APIs

SUCCESS_READER_OP( 0 )
Successful operation. No Error Found.
#Handled by the DLL. The DLL has to do the consistent checking even a “Success Response Status” is returned by the device.

ERR_INTERNAL_UNEXPECTED(-1000)
Library internal unexpected error.
#Handled by the DLL

ERR_PORT_INVALID(-2000)
The port is invalid.
#Handled by the DLL

ERR_PORT_OCCUPIED(-2010)
The port is occupied by another application.
#Handled by the DLL

ERR_HANDLE_INVALID(-2020)
The handle is invalid.
#Handled by the DLL

ERR_INCORRECT_PARAM(-2030)
Incorrect Parameter.
#Handled by the DLL

ERR_READER_NO_TAG(-3000, or 0xF448)
No TAG in reachable range / selected.
#Corresponding to the << Response Status ‘N’ >>.

ERR_READER_OP_FAILURE(-3030, or 0xF42A)
Operation failed.
#Corresponding to the << Response Status ‘F’ >>.

ERR_READER_UNKNOWN(-3040, or 0xF420)
Reader unknown error.
#Corresponding to the << Response Status ‘C’, ‘O’, ‘X’ & ‘?’ >>.
APPENDIX:

1. Error Codes returned by High Level APIs (Cont.)

ERR_READER_LOGIN_INVALID_STORED_KEY_FORMAT(-4010, or 0xF056)
Invalid stored key format in login process.
#Handled by the DLL.

ERR_READER_LOGIN_FAIL(-4011, or 0xF055)
Login failed.
#Corresponding to the << Response Status ‘I’ >>.

ERR_READER_OP_AUTH_FAIL(-4012, or 0xF054)
The operation or access is not authorized.
#Corresponding to the << Response Status ‘I’ >>.

ERR_READER_VALUE_DEC_EMPTY(-4030, or 0xF042)
Decrement failure (empty).
#Corresponding to the << Response Status ‘E’ >>.

ERR_READER_VALUE_INC_OVERFLOW(-4031, or 0xF041)
Increment Overflow.
#Corresponding to the << Response Status ‘E’ >>.

ERR_READER_VALUE_OP_FAILURE(-4032, 0xF040)
Value Operations failure. E.g. Value Increment
#Corresponding to the << Response Status ‘I’ >>.

ERR_READER_VALUE_INVALID_BLOCK(-4033, 0xF03F)
Block doesn’t contain value.
#Corresponding to the << Response Status ‘F’ >>.

ERR_READER_VALUE_ACCESS_FAILURE(-4034, 0xF03E)
Value Access failure.
#Corresponding to the << Response Status ‘U’ >>.
APPENDIX:

2. Possible TAG Types

<table>
<thead>
<tr>
<th>TAG Type Value</th>
<th>TAG Type Description</th>
<th>TAG SN Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01</td>
<td>Mifare Light</td>
<td>4</td>
</tr>
<tr>
<td>0x02</td>
<td>Mifare 1K</td>
<td>4</td>
</tr>
<tr>
<td>0x03</td>
<td>Mifare 4K</td>
<td>4</td>
</tr>
<tr>
<td>0x04</td>
<td>Mifare DESFire</td>
<td>7</td>
</tr>
<tr>
<td>0x05</td>
<td>Mifare Ultralight</td>
<td>7</td>
</tr>
<tr>
<td>0x06</td>
<td>JCOP30</td>
<td>4</td>
</tr>
<tr>
<td>0x07</td>
<td>Shanghai Transport</td>
<td>4</td>
</tr>
<tr>
<td>0x08</td>
<td>MPCOS Combi</td>
<td>4</td>
</tr>
<tr>
<td>0x80</td>
<td>ISO Type B, Calypso</td>
<td>4</td>
</tr>
<tr>
<td>0x81</td>
<td>ASK CTS256B, Type B</td>
<td>8</td>
</tr>
<tr>
<td>0x82</td>
<td>ASK CTS512B, Type B</td>
<td>8</td>
</tr>
</tbody>
</table>

#The TAG SN Format of ASK CTS256B and CTS512B Cards

<table>
<thead>
<tr>
<th>1st Byte</th>
<th>2nd Byte</th>
<th>3rd Byte</th>
<th>4th Byte</th>
<th>5th to 8th Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturing Code</td>
<td>Product Code</td>
<td>Embedded Code</td>
<td>Application Code</td>
<td>MSB (H)</td>
</tr>
<tr>
<td>XX</td>
<td>0x50 (CTS256B) or 0x60 (CTS512B)</td>
<td>XX</td>
<td>XX</td>
<td>XX</td>
</tr>
</tbody>
</table>

3. USB ID and Drivers for ACR120U

- VID_0x072F & PID_0x8003 as the USB ID of ACR120U
- ACR120.SYS will be used as the driver name for ACR120U based on ST7263
- ACR120U.DLL will be used as the DLL name for ACR120U based on ST7263.
APPENDIX:

4. Standard Program Flow

1) Before executing any Card Commands, get the Reader Handle first.
2) Select a TAG
3) Login the TAG
4) Access the TAG
5) Close the Reader Handle

// ACR120_Sample.c; a very simple program for accessing Philips MIFare 1K Tags

#include “acr120.h”

void main(void)
{
    INT16 hReader = -1;
    UINT8 Length, SN[10], Data[16], Type;

    // Get the Reader Handle first. Open a communication channel (USB Interface)
    hReader=ACR120_Open(ACR120_USB1);
    if(hReader<0){ // error happened!!! };

    // Assume the Reader Handle is ready, then “Select a TAG”
    ACR120_Select(hReader, &Type, &Length, SN);

    // Assume a TAG is selected, then “Login Sector 0x02” using “Default Key F”
    ACR120_Login(hReader, 0x02, AC_MIFARE_LOGIN_KEYTYPE_DEFAULT_F, 0, NULL);

    // Assume the Sector is authorized, then “Read data from Block 0x08 of Sector 0x02”
    ACR120_Read(hReader, 0x08, Data);

    /*
     Some operations.
     */

    ACR120_Close(hReader); // Close the port and quit the program
    return;
}
APPENDIX:

5. Physical and Logical Block/Sector Calculation

1. Mifare 1K
   • Logical Sector is equal to Physical sector, which are 0 to 15.
   • Logical block of each sector is from 0 to 3.
   • Physical blocks = ((Sector * 4) + Logical block)

2. Mifare 4K
   • Case 1: If \{0 <= Logical Sector <= 31\}
     ▪ Physical sector is equal to Logical.
     ▪ Logical block of each sector is from 0 to 3.
     ▪ Physical blocks = ((Sector * 4) + Logical block)
   • Case 2: If \{32 <= Logical Sector <= 39\}
     ▪ Physical Sector = Logical Sector + ((Logical Sector - 32) * 3)
     ▪ Logical block of each sector is from 0 to 15.
     ▪ Physical blocks = ((Logical Sector - 32) * 16) + 128 + Logical block
LAMPIRAN C

Datasheet Mifare 1K
## CONTENTS

1 FEATURES.......................................................................................................................... 4  
1.1 MIFARE® RF Interface (ISO/IEC 14443 A)................................................................. 4  
1.2 EEPROM.......................................................................................................................... 4  
1.3 Security ............................................................................................................................ 4  
2 GENERAL DESCRIPTION................................................................................................... 5  
2.1 Contactless Energy and Data Transfer ......................................................................... 5  
2.2 Anticollision ...................................................................................................................... 5  
2.3 User Convenience ........................................................................................................... 5  
2.4 Security ............................................................................................................................ 5  
2.5 Multi-application Functionality ....................................................................................... 5  
2.6 Delivery Options ............................................................................................................ 6  
3 FUNCTIONAL DESCRIPTION.............................................................................................. 6  
3.1 Block Description .......................................................................................................... 6  
3.2 Communication Principle .............................................................................................. 7  
3.2.1 REQUEST STANDARD / ALL ...................................................................................... 7  
3.2.2 ANTICOLLISION LOOP .............................................................................................. 7  
3.2.3 SELECT CARD .............................................................................................................. 7  
3.2.4 3 PASS AUTHENTICATION ....................................................................................... 7  
3.2.5 MEMORY OPERATIONS ........................................................................................... 8  
3.3 Data Integrity ................................................................................................................... 8  
3.4 Security ............................................................................................................................ 8  
3.4.1 THREE PASS AUTHENTICATION SEQUENCE ....................................................... 8  
3.5 RF Interface ..................................................................................................................... 8  
3.6 Memory Organisation ...................................................................................................... 9  
3.6.1 MANUFACTURER BLOCK ....................................................................................... 10  
3.6.2 DATA BLOCKS ......................................................................................................... 10  
3.6.3 SECTOR TRAILER (BLOCK 3) .................................................................................. 11  
3.7 Memory Access .............................................................................................................. 12  
3.7.1 ACCESS CONDITIONS ......................................................................................... 13  
3.7.2 ACCESS CONDITIONS FOR THE SECTOR TRAILER ....................................... 13  
3.7.3 ACCESS CONDITIONS FOR DATA BLOCKS ...................................................... 15  
4 DEFINITIONS .................................................................................................................... 16
Functional Specification

Standard Card IC MF1 IC S50

5 LIFE SUPPORT APPLICATIONS .......................................................................................... 16
6 REVISION HISTORY ........................................................................................................... 17

MIFARE® is a registered trademark of Philips Electronics N.V.
1 FEATURES

1.1 MIFARE® RF Interface (ISO/IEC 14443 A)

- Contactless transmission of data and supply energy (no battery needed)
- Operating distance: Up to 100mm (depending on antenna geometry)
- Operating frequency: 13.56 MHz
- Fast data transfer: 106 kbit/s
- High data integrity: 16 Bit CRC, parity, bit coding, bit counting
- True anticollision
- Typical ticketing transaction: < 100 ms (including backup management)

1.2 EEPROM

- 1 Kbyte, organized in 16 sectors with 4 blocks of 16 bytes each (one block consists of 16 byte)
- User definable access conditions for each memory block
- Data retention of 10 years.
- Write endurance 100,000 cycles

1.3 Security

- Mutual three pass authentication (ISO/IEC DIS9798-2)
- Data encryption on RF-channel with replay attack protection
- Individual set of two keys per sector (per application) to support multi-application with key hierarchy
- Unique serial number for each device
- Transport key protects access to EEPROM on chip delivery
2 GENERAL DESCRIPTION

Philips has developed the MIFARE® MF1 IC S50 to be used in contactless smart cards according to ISO/IEC 14443A. The communication layer (MIFARE® RF Interface) complies to parts 2 and 3 of the ISO/IEC 14443A standard. The security layer sports the field-proven CRYPTO1 stream cipher for secure data exchange of the MIFARE® Classic family.

2.1 Contactless Energy and Data Transfer

In the MIFARE® system, the MF1 IC S50 is connected to a coil with a few turns and then embedded in plastic to form the passive contactless smart card. No battery is needed. When the card is positioned in the proximity of the Read Write Device (RWD) antenna, the high speed RF communication interface allows to transmit data with 106 kBit/s.

2.2 Anticollision

An intelligent anticollision function allows to operate more than one card in the field simultaneously. The anticollision algorithm selects each card individually and ensures that the execution of a transaction with a selected card is performed correctly without data corruption resulting from other cards in the field.

2.3 User Convenience

The MIFARE® system is designed for optimal user convenience. The high data transmission rate for example allows complete ticketing transactions to be handled in less than 100 ms. Thus, the MIFARE® card user is not forced to stop at the RWD antenna leading to a high throughput at gates and reduced boarding times onto busses. The MIFARE® card may also remain in the wallet during the transaction, even if there are coins in it.

2.4 Security

Special emphasis has been placed on security against fraud. Mutual challenge and response authentication, data ciphering and message authentication checks protect the system from any kind of tampering and thus make it attractive for ticketing applications. Serial numbers, which can not be altered, guarantee the uniqueness of each card.

2.5 Multi-application Functionality

The MIFARE® system offers real multi-application functionality comparable to the features of a processor card. Two different keys for each sector support systems using key hierarchies.
2.6 Delivery Options

- Die on wafer
- Bumped die on wafer
- Chip Card Module

3 FUNCTIONAL DESCRIPTION

3.1 Block Description

The MF1 IC S50 chip consists of the 1 Kbyte EEPROM, the RF-Interface and the Digital Control Unit. Energy and data are transferred via an antenna, which consists of a coil with a few turns directly connected to the MF1 IC S50. No further external components are necessary. (For details on antenna design please refer to the document MIFARE® Card IC Coil Design Guide.)

- RF-Interface:
  - Modulator/Demodulator
  - Rectifier
  - Clock Regenerator
  - Power On Reset
  - Voltage Regulator
- Anticollision: Several cards in the field may be selected and operated in sequence
- Authentication: Preceding any memory operation the authentication procedure ensures that access to a block is only possible via the two keys specified for each block
- Control & Arithmetic Logic Unit: Values are stored in a special redundant format and can be incremented and decremented
- EEPROM-Interface
- Crypto unit: The field-proven CRYPTO1 stream cipher of the MIFARE® Classic family ensures a secure data exchange
- EEPROM: 1 Kbyte are organized in 16 sectors with 4 blocks each. A block contains 16 bytes. The last block of each sector is called “trailer”, which contains two secret keys and programmable access conditions for each block in this sector.
3.2 Communication Principle

The commands are initiated by the RWD and controlled by the Digital Control Unit of the MF1 IC S50 according to the access conditions valid for the corresponding sector.

3.2.1 REQUEST STANDARD / ALL

After Power On Reset (POR) of a card it can answer to a request command - sent by the RWD to all cards in the antenna field - by sending the answer to request code (ATQA according to ISO/IEC 14443A).

3.2.2 ANTICOLLISION LOOP

In the anticollision loop the serial number of a card is read. If there are several cards in the operating range of the RWD, they can be distinguished by their unique serial numbers and one can be selected (select card) for further transactions. The unselected cards return to the standby mode and wait for a new request command.

3.2.3 SELECT CARD

With the select card command the RWD selects one individual card for authentication and memory related operations. The card returns the Answer To Select (ATS) code (= 08h), which determines the type of the selected card. Please refer to the document MIFARE® Standardised Card Type Identification Procedure for further details.

3.2.4 3 PASS AUTHENTICATION

After selection of a card the RWD specifies the memory location of the following memory access and uses the corresponding key for the 3 pass authentication procedure. After a successful authentication all memory operations are encrypted.

---

**Transaction Sequence**

- **POR**
- **Request Standard**
- **Request All**
- **Anticollision Loop**
  - Get Serial Number
- **Select Card**
- **3 Pass Authentication**
  - sector specific
- **Read Block**
- **Write Block**
- **Decrement**
- **Increment**
- **Restore**
- **Halt**
- **Transfer**

**Typical Transaction Time**

- **Identification and Selection Procedure**
  - 3 ms without collision
  - + 1 ms for each collision

- **Authentication Procedure**
  - 2 ms

- **Memory Operations**
  - 2.5 ms read block
  - 6.0 ms write block
3.2.5 MEMORY OPERATIONS

After authentication any of the following operations may be performed:

- Read block
- Write block
- Decrement: Decrements the contents of a block and stores the result in a temporary internal data-register
- Increment: Increments the contents of a block and stores the result in the data-register
- Restore: Moves the contents of a block into the data-register
- Transfer: Writes the contents of the temporary internal data-register to a value block

3.3 Data Integrity

Following mechanisms are implemented in the contactless communication link between RWD and card to ensure very reliable data transmission:

- 16 bits CRC per block
- Parity bits for each byte
- Bit count checking
- Bit coding to distinguish between "1", "0", and no information
- Channel monitoring (protocol sequence and bit stream analysis)

3.4 Security

To provide a very high security level a three pass authentication according to ISO 9798-2 is used.

3.4.1 THREE PASS AUTHENTICATION SEQUENCE

a) The RWD specifies the sector to be accessed and chooses key A or B.
b) The card reads the secret key and the access conditions from the sector trailer. Then the card sends a random number as the challenge to the RWD (pass one).
c) The RWD calculates the response using the secret key and additional input. The response, together with a random challenge from the RWD, is then transmitted to the card (pass two).
d) The card verifies the response of the RWD by comparing it with its own challenge and then it calculates the response to the challenge and transmits it (pass three).
e) The RWD verifies the response of the card by comparing it to its own challenge.

After transmission of the first random challenge the communication between card and RWD is encrypted.

3.5 RF Interface

The RF-interface is according to the standard for contactless smart cards ISO/IEC 14443A.

The carrier field from the RWD is always present (with short pauses when transmitting), because it is used for the power supply of the card.

For both directions of data communication there is only one start bit at the beginning of each frame. Each byte is transmitted with a parity bit (odd parity) at the end. The LSB of the byte with the lowest address of the selected block is transmitted first. The maximum frame length is 163 bits (16 data bytes + 2 CRC bytes = 16 * 9 + 2 * 9 + 1 start bit).
3.6 Memory Organisation

The 1024 x 8 bit EEPROM memory is organized in 16 sectors with 4 blocks of 16 bytes each.

In the erased state the EEPROM cells are read as a logical “0”, in the written state as a logical “1”.

<table>
<thead>
<tr>
<th>Sector</th>
<th>Block</th>
<th>Description</th>
<th>Byte Number within a Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>3</td>
<td>Sector Trailer 15</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>Sector Trailer 14</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>Sector Trailer 1</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>Sector Trailer 0</td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Data</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Manufacturer Block</td>
<td></td>
</tr>
</tbody>
</table>
3.6.1 MANUFACTURER BLOCK

This is the first data block (block 0) of the first sector (sector 0). It contains the IC manufacturer data. Due to security and system requirements this block is write protected after having been programmed by the IC manufacturer at production.

![Manufacturer Block Diagram]

3.6.2 DATA BLOCKS

All sectors contain 3 blocks of 16 bytes for storing data (Sector 0 contains only two data blocks and the read-only manufacturer block).

The data blocks can be configured by the access bits as

- read/write blocks for e.g. contactless access control or
- value blocks for e.g. electronic purse applications, where additional commands like increment and decrement for direct control of the stored value are provided.

An authentication command has to be carried out before any memory operation in order to allow further commands.

3.6.2.1 Value Blocks

The value blocks allow to perform electronic purse functions (valid commands: read, write, increment, decrement, restore, transfer). The value blocks have a fixed data format which permits error detection and correction and a backup management.

A value block can only be generated through a write operation in the value block format:

- Value: Signifies a signed 4-byte value. The lowest significant byte of a value is stored in the lowest address byte. Negative values are stored in standard 2’s complement format. For reasons of data integrity and security, a value is stored three times, twice non-inverted and once inverted.
- Adr: Signifies a 1-byte address, which can be used to save the storage address of a block, when implementing a powerful backup management. The address byte is stored four times, twice inverted and non-inverted. During increment, decrement, restore and transfer operations the address remains unchanged. It can only be altered via a write command.

<table>
<thead>
<tr>
<th>Byte Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Value</td>
</tr>
<tr>
<td>1</td>
<td>Value</td>
</tr>
<tr>
<td>2</td>
<td>Value</td>
</tr>
<tr>
<td>3</td>
<td>Adr</td>
</tr>
<tr>
<td>4</td>
<td>Adr</td>
</tr>
<tr>
<td>5</td>
<td>Adr</td>
</tr>
<tr>
<td>6</td>
<td>Adr</td>
</tr>
<tr>
<td>7</td>
<td>Adr</td>
</tr>
<tr>
<td>8</td>
<td>Value</td>
</tr>
<tr>
<td>9</td>
<td>Value</td>
</tr>
<tr>
<td>10</td>
<td>Value</td>
</tr>
<tr>
<td>11</td>
<td>Adr</td>
</tr>
<tr>
<td>12</td>
<td>Adr</td>
</tr>
<tr>
<td>13</td>
<td>Adr</td>
</tr>
<tr>
<td>14</td>
<td>Adr</td>
</tr>
<tr>
<td>15</td>
<td>Adr</td>
</tr>
</tbody>
</table>
3.6.3 SECTOR TRAILER (BLOCK 3)

Each sector has a sector trailer containing the

- secret keys A and B (optional), which return logical “0”s when read and
- the access conditions for the four blocks of that sector, which are stored in bytes 6...9. The access bits also specify the type (read/write or value) of the data blocks.

If key B is not needed, the last 6 bytes of block 3 can be used as data bytes.

Byte 9 of the sector trailer is available for user data. For this byte apply the same access rights as for byte 6, 7 and 8.

<table>
<thead>
<tr>
<th>Byte Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Key A</td>
</tr>
<tr>
<td>1</td>
<td>Access Bits</td>
</tr>
<tr>
<td>2</td>
<td>Key B (optional)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.7 Memory Access

Before any memory operation can be carried out, the card has to be selected and authenticated as described previously.

The possible memory operations for an addressed block depend on the key used and the access conditions stored in the associated sector trailer.

<table>
<thead>
<tr>
<th>Memory Operations</th>
<th>Description</th>
<th>Valid for Block Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>reads one memory block</td>
<td>read/write, value and sector trailer</td>
</tr>
<tr>
<td>Write</td>
<td>writes one memory block</td>
<td>read/write, value and sector trailer</td>
</tr>
<tr>
<td>Increment</td>
<td>increments the contents of a block and stores the result in the internal data register</td>
<td>value</td>
</tr>
<tr>
<td>Decrement</td>
<td>decrements the contents of a block and stores the result in the internal data register</td>
<td>value</td>
</tr>
<tr>
<td>Transfer</td>
<td>writes the contents of the internal data register to a block</td>
<td>value</td>
</tr>
<tr>
<td>Restore</td>
<td>reads the contents of a block into the internal data register</td>
<td>value</td>
</tr>
</tbody>
</table>
### 3.7.1 ACCESS CONDITIONS

The access conditions for every data block and sector trailer are defined by 3 bits, which are stored non-inverted and inverted in the sector trailer of the specified sector.

The access bits control the rights of memory access using the secret keys A and B. The access conditions may be altered, provided one knows the relevant key and the current access condition allows this operation.

<table>
<thead>
<tr>
<th>Access Bits</th>
<th>Valid Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1, C2, C3</td>
<td>read, write</td>
</tr>
<tr>
<td>C1, C2, C3</td>
<td>read, write, increment, decrement, transfer, restore</td>
</tr>
<tr>
<td>C1, C2, C3</td>
<td>read, write, increment, decrement, transfer, restore</td>
</tr>
<tr>
<td>C1, C2, C3</td>
<td>read, write, increment, decrement, transfer, restore</td>
</tr>
</tbody>
</table>

Note: In the following description the access bits are mentioned in the non-inverted mode only.

The internal logic of the MF1 IC S50 ensures that the commands are executed only after an authentication procedure or never.

### 3.7.2 ACCESS CONDITIONS FOR THE SECTOR TRAILER

Depending on the access bits for the sector trailer (block 3) the read/write access to the keys and the

<table>
<thead>
<tr>
<th>Byte Number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access Bits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Key B (optional)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
access bits is specified as ‘never’, ‘key A’, ‘key B’ or
key A|B’ (key A or key B).

On chip delivery the access conditions for the sector
configuration, new cards must be authenticated with
key A.

Since the access bits themselves can also be
blocked, special care should be taken during
personalization of cards.

Note: the grey marked lines are access conditions
where key B is readable and may be used for data.

<table>
<thead>
<tr>
<th>Access bits</th>
<th>Access condition for KEYA</th>
<th>Access condition for KEYB</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1  C2  C3</td>
<td>read  write  read  write</td>
<td>read  write  read  write</td>
<td></td>
</tr>
<tr>
<td>0  0  0</td>
<td>never  key A  key A  never</td>
<td>never  key A  key A  Key B may be read</td>
<td></td>
</tr>
<tr>
<td>0  1  0</td>
<td>never  never  key A  never</td>
<td>never  key A  never  Key B may be read</td>
<td></td>
</tr>
<tr>
<td>1  0  0</td>
<td>never  key B  key A</td>
<td>B  never</td>
<td>never  never  Key B may be read</td>
</tr>
<tr>
<td>1  1  0</td>
<td>never  never  key A</td>
<td>B  never</td>
<td>never  never</td>
</tr>
<tr>
<td>0  0  1</td>
<td>never  key A  key A  key A  key A  Key B may be read, transport configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0  1  1</td>
<td>never  key B  key A</td>
<td>B  key B  never  key B</td>
<td></td>
</tr>
<tr>
<td>1  0  1</td>
<td>never  never  key A</td>
<td>B  key B  never  never</td>
<td></td>
</tr>
<tr>
<td>1  1  1</td>
<td>never  never  key A</td>
<td>B  never  never  never</td>
<td></td>
</tr>
</tbody>
</table>
3.7.3 ACCESS CONDITIONS FOR DATA BLOCKS

Depending on the access bits for data blocks (blocks 0..2) the read/write access is specified as ‘never’, ‘key A’, ‘key B’ or ‘key A|B’ (key A or key B). The setting of the relevant access bits defines the application and the corresponding applicable commands.

- Read/write block: The operations read and write are allowed.
- Value block: Allows the additional value operations increment, decrement, transfer und restore. In one case (‘001’) only read and decrement are possible for a non-rechargeable card. In the other case (‘110’) recharging is possible by using key B.
- Manufacturer block: The read-only condition is not affected by the access bits setting!
- Key management: In transport configuration key A must be used for authentication¹.

<table>
<thead>
<tr>
<th>Access bits</th>
<th>Access condition for</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1 C2 C3</td>
<td>read</td>
<td>write</td>
</tr>
<tr>
<td>0 0 0</td>
<td>key A</td>
<td>B¹</td>
</tr>
<tr>
<td>0 1 0</td>
<td>key A</td>
<td>B¹</td>
</tr>
<tr>
<td>1 0 0</td>
<td>key A</td>
<td>B¹</td>
</tr>
<tr>
<td>1 1 0</td>
<td>key A</td>
<td>B¹</td>
</tr>
<tr>
<td>0 0 1</td>
<td>key A</td>
<td>B¹</td>
</tr>
<tr>
<td>0 1 1</td>
<td>key B¹</td>
<td>never</td>
</tr>
<tr>
<td>1 0 1</td>
<td>key B¹</td>
<td>never</td>
</tr>
<tr>
<td>1 1 1</td>
<td>never</td>
<td>never</td>
</tr>
</tbody>
</table>

¹ if Key B may be read in the corresponding Sector Trailer it cannot serve for authentication (all grey marked lines in previous table). **Consequences:** If the RWD tries to authenticate any block of a sector with key B using grey marked access conditions, the card will refuse any subsequent memory access after authentication.
4 DEFINITIONS

Data sheet status

<table>
<thead>
<tr>
<th>Specification</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective specification</td>
<td>This data sheet contains target or goal specifications for product development.</td>
</tr>
<tr>
<td>Preliminary specification</td>
<td>This data sheet contains preliminary data; supplementary data may be published later.</td>
</tr>
<tr>
<td>Product specification</td>
<td>This data sheet contains final product specifications.</td>
</tr>
</tbody>
</table>

Limiting values

Limiting values given are in accordance with the Absolute Maximum Rating System (IEC 134). Stress above one or more of the limiting values may cause permanent damage to the device. These are stress ratings only and operation of the device at these or at any other conditions above those given in the Characteristics section of the specification is not implied. Exposure to limiting values for extended periods may affect device reliability.

Application information

Where application information is given, it is advisory and does not form part of the specification.

5 LIFE SUPPORT APPLICATIONS

These products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. Philips customers using or selling these products for use in such applications do so on their own risk and agree to fully indemnify Philips for any damages resulting from such improper use or sale.
### 6 REVISION HISTORY

**Table 1** Functional Specification MF1 IC S50 Revision History

<table>
<thead>
<tr>
<th>REVISION</th>
<th>DATE</th>
<th>CPCN</th>
<th>PAGE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>0501</td>
<td>2001</td>
<td>9, 10</td>
<td>New Coding Manufacturer Block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>05013</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5.0</td>
<td>1199</td>
<td></td>
<td></td>
<td>New Layout: Revised. Includes MF1 IC S50 05 silicon.</td>
</tr>
<tr>
<td>1.0</td>
<td></td>
<td>-</td>
<td></td>
<td>Initial version.</td>
</tr>
</tbody>
</table>
LAMPIRAN D

Spesifikasi Ethernet to Serial Gateway (EGSR-7150MJ)
If you have any question about WIZnet Products, write them down onto our Q&A Board on our website at [www.wiznet.co.kr](http://www.wiznet.co.kr). A WIZnet engineer will promptly provide you with an answer.
1. Introduction .................................................................................................................3
  1.1. Key Features ........................................................................................................3
  1.2. Products Contents (EG-SR-7150MJ-EVB) ..........................................................3
  1.3. Specifications of the EG-SR-7150MJ .................................................................6
2. Getting Started .........................................................................................................7
  2.1. Hardware Installation procedure ........................................................................7
  2.2. Configuration Tool ...............................................................................................8
    2.2.1. Configuration tool features .........................................................................8
  2.3. Serial Communication Specification .................................................................12
    2.3.1. Frame Format ..............................................................................................12
    2.3.2. STX & ETX .................................................................................................12
    2.3.3. Reply Code ................................................................................................13
    2.3.4. Command Code .........................................................................................13
  2.4. Operation Flow ....................................................................................................15
  2.5. Factory Default ....................................................................................................15
3. Demonstration and Test ...........................................................................................16
  3.1. Case 1: Getting IP address using H/W trigger ..................................................16
  3.2. Case 2: Changing IP address using H/W trigger .............................................16
  3.3. Case 3: Changing IP address using S/W trigger .............................................17
4. PIN Assignment and Dimensions ........................................................................18
5. Reference Schematic ..............................................................................................20
6. ETC ..........................................................................................................................20
  6.1. Firmware Uploading through the FLIP software ..............................................20
  6.2. Warranty .............................................................................................................21
# 1. Introduction

The EG-SR-7150MJ is a gateway module that converts serial data into TCP/IP data type. It transmits the data sent by a serial equipment to the Internet and TCP/IP data to the equipment.

With the EG-SR-7150MJ mounted with an RJ-45 connector, users can have an easier and quicker interface with the Ethernet.

The EG-SR-7150MJ provides serial commands, with which the developers of any serial device can add local configuration capability to their products. For example, a card reader developer can program the keypad on a card reader to configure serial or network on-site without the use of a laptop or PC.

## 1.1. Key Features

- Ready-to-go serial to Ethernet gateway module mounted with an RJ-45 connector
- Serial Command Support
  - Simple command frame format
  - Comprehensive & readable command set for network and serial settings
  - On-site configuration without PC
- High stability & reliability by using a W3150A WIZnet Chip, a fully-hardwired TCP/IP stack
- Easy and powerful configuration program
- 10/100Mbps Ethernet interface, Max. 230Kbps Serial interface
- RoHS compliant

## 1.2. Products Contents (EG-SR-7150MJ-EVB)

The EG-SR-7150MJ-EVB, the evaluation kit for the EG-SR-7150MJ contains the following items;
<table>
<thead>
<tr>
<th>Test Board for EG-SR-7150MJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>12pin Cable</td>
</tr>
<tr>
<td>(Connecting EG-SR-7150MJ to Test Board)</td>
</tr>
<tr>
<td>Serial Cable</td>
</tr>
<tr>
<td>(Connecting Serial Device to Test Board)</td>
</tr>
<tr>
<td>LAN Cable</td>
</tr>
<tr>
<td>(Connecting EG-SR-7150MJ to Host)</td>
</tr>
<tr>
<td>Power</td>
</tr>
<tr>
<td>(DC 5V Adaptor)</td>
</tr>
</tbody>
</table>
☞ Please immediately notify your sales representative if any of the items above is missing or damaged.
### 1.3. Specifications of the EG-SR-7150MJ

<table>
<thead>
<tr>
<th>Category</th>
<th>Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Form Factor</td>
<td>2mm Pitch 2x6 pins, 62x40 mm</td>
</tr>
<tr>
<td>LAN Interface</td>
<td>10/100 Mbps auto-sensing, RJ-45 connector</td>
</tr>
<tr>
<td>Protocol</td>
<td>TCP, UDP, IP, ARP, ICMP, MAC, (IGMP, PPPoE)</td>
</tr>
<tr>
<td>CPU</td>
<td>AT89C51RC2 (8bit MCU and 32K Flash)</td>
</tr>
<tr>
<td>Serial Interface</td>
<td>RS 232 (LVTTL)</td>
</tr>
<tr>
<td>Serial Signals</td>
<td>TXD, RXD, RTS, CTS, GND</td>
</tr>
<tr>
<td>Serial Parameters</td>
<td>Parity : None, Even, Odd</td>
</tr>
<tr>
<td></td>
<td>Data bits : 7, 8</td>
</tr>
<tr>
<td></td>
<td>Flow control : RTS/CTS, XON/XOFF</td>
</tr>
<tr>
<td></td>
<td>Speed : up to 230Kbps</td>
</tr>
<tr>
<td>Management</td>
<td>Configuration utility based on Windows</td>
</tr>
<tr>
<td>Temperature</td>
<td>0°C<del>70°C (Operating), -40°C</del>85°C (Storage)</td>
</tr>
<tr>
<td>Humidity</td>
<td>10~90%</td>
</tr>
<tr>
<td>Power</td>
<td>150mA @ 3.3V (max)</td>
</tr>
<tr>
<td>Size</td>
<td>40mm x 62mm x 17mm</td>
</tr>
</tbody>
</table>
2. Getting Started

This Chapter describes how to set up and configure the EG-SR-7150MJ.

The following items are required to get started.

- Power Cable (included in the EG-SR-7150MJ-EVB package)
- Serial and Ethernet Cables (included in the of EG-SR-7150MJ-EVB package)
- PC or Laptop with Network Interface Card (hereafter, NIC) and/or one RS232 serial port

2.1. Hardware Installation procedure

Follow steps below to prepare the module and evaluation board for testing.

**STEP 1**: Connect the EG-SR-7150MJ module to the test board by using the 12pin cable.

**STEP 2**: Connect the 5V DC power line to the power jack of the test board.

**STEP 3**: Use the RJ45 Ethernet cable in order to connect the module to an Ethernet network.

**STEP 4**: Use the serial data cable to connect the test board to a serial device.
2.2. Configuration Tool

2.2.1. Configuration tool features

① Search
The Search function is used to search all modules existing on the same Subnet. The UDP broadcast is used for searching modules on a LAN. The MAC address for a searched module will be listed in the "Module list".

If Direct IP Search is checked, TCP will be used for searching instead of UDP. This mode is used more for searching the EG-SR-7150MJ modules on remote networks than local networks with the same subnet. An IP address assigned to the module will be required.
② Setting
If you select one of the MAC addresses listed in the “Module list”, the configuration value of
the selected module will be displayed. After changing each value in the configuration program,
click the “Setting” button to complete the configuration.
The module will be initialized with the new configurations.

③ IP Configuration method: Static, DHCP
Static: The IP address can be manually assigned by users.
DHCP: The module assigns IP, subnet and gateway addresses by acquiring them from the
DHCP server
☞ Other configurations should be set manually except for the IP configuration of DHCP.

④ Operation mode: TCP server, TCP client, UDP
Three different operation modes are supported — TCP Server, TCP Client, and UDP.
The main difference between the TCP and UDP protocols is that TCP guarantees the delivery of
data by requesting the recipient to send an acknowledgement to the sender. On the other
hand, UDP does not require this type of verification, so data can be delivered quicker, but its
delivery can not be guaranteed.
The TCP Server and TCP Client mode are related to the first step of connection establishment.
Once the connection is established, data will be transparently transmitted in both directions
(from Server to Client or from Client to Server).

TCP server mode
To operate this mode, the **Local IP, Subnet, gateway address and local port number** should be configured. The EG-SR-7150MJ waits to be connected by the host computer, allowing the host computer to establish a connection and get data from the serial device.

As illustrated in the figure above, the data transmission is as follows:

1. The host connects to the EG-SR-7150MJ which is configured as TCP Server Mode.
2. Once the connection is established, data can be transmitted in both directions - from the host to the EG-SR-7150MJ, and from the EG-SR-7150MJ to the host.

**TCP client mode**

To operate this mode, the **Local IP, Subnet, gateway address, server IP, server port number** should be set. In the **TCP Client mode**, the EG-SR-7150MJ proceeds active open for establishing a TCP connection to a host computer.

As illustrated in the figure above, data transmission is as follows:

1. The EG-SR-7150MJ operating as TCP Client Mode establishes a connection based on the condition set in the **TCP client connection method** (**Startup, Any character**). i.e. the EG-SR-7150MJ can try to connect as soon as one starts up(**Startup**), or later when data from serial device arrives. (**Any character**).
2. After the connection is established, data can be transmitted in both directions - from the host to the EG-SR-7150-MJ, and from the EG-SR-7150-MJ to the host.

**UDP mode**
In UDP mode, any TCP/IP connection procedure is not required.

⑤ **Serial command method: H/W trigger, S/W trigger**
With this menu, you can designate how the Serial command mode can be entered. Two types are supported - H/W Trigger and S/W Trigger.

**H/W trigger:** Serial command mode can be triggered by pulling H/W trigger pin to low. It can be exited by pulling it to high.

**S/W trigger:** Serial command mode can be triggered when 3 user-defined characters are detected. It can be exited by using the WR command.
Delimiter: Time, Size, Character

You can designate how the serial data can be packed and sent to the Ethernet. There are 3 delimiters - Time, Size and Character. If all of them are set as '0', whenever the serial data arrives, they will be sent to the Ethernet without any condition. When any of the three delimiters is satisfied, data can be sent to the Ethernet.

Ex) Delimiter: Size=10, Char=0x0D
    Serial data: 0123456789abc
    Ethernet data: 0123456789
☞ “abc” data remains in the serial buffer of module

Inactivity time

After the connection is established, if there is not any data transmission within the time defined in "Inactivity time", the connection will be automatically closed.

Upload

Upload the firmware through the network.
☞ After uploading the firmware, 10~20 seconds are required for initialization.

2.3. Serial Communication Specification

In this chapter, we describe the structure of the data frames used in issuing commands and receiving responses to and from the device.

2.3.1. Frame Format

Command Frame format

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>STX</th>
<th>Command code</th>
<th>Parameter</th>
<th>ETX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length(bytes)</td>
<td>1</td>
<td>2</td>
<td>Variable</td>
<td>1</td>
</tr>
</tbody>
</table>

Reply Frame format

<table>
<thead>
<tr>
<th>Descriptor</th>
<th>STX</th>
<th>Reply code</th>
<th>Parameter</th>
<th>ETX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length(bytes)</td>
<td>1</td>
<td>1</td>
<td>Variable</td>
<td>1</td>
</tr>
</tbody>
</table>

2.3.2. STX & ETX

<table>
<thead>
<tr>
<th>Setting</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>STX</td>
<td>‘&lt;’ : Hex = 3Ch</td>
</tr>
<tr>
<td>ETX</td>
<td>‘&gt;’ : Hex = 3Eh</td>
</tr>
</tbody>
</table>
### 2.3.3. Reply Code

<table>
<thead>
<tr>
<th>Reply</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Command was successful</td>
</tr>
<tr>
<td>F</td>
<td>Command failed</td>
</tr>
<tr>
<td>1</td>
<td>Invalid command</td>
</tr>
<tr>
<td>2</td>
<td>Invalid parameter</td>
</tr>
<tr>
<td>E</td>
<td>Enter serial command mode</td>
</tr>
</tbody>
</table>

### 2.3.4. Command Code

<table>
<thead>
<tr>
<th>Command</th>
<th>Parameter</th>
<th>Comments</th>
</tr>
</thead>
</table>
| WI      | xxx.xxx.xxx.xxx  
(eg. 192.168.11.133) | Set Local IP |
| WS      | xxx.xxx.xxx.xxx  
(eg. 255.255.255.0) | Set Subnet mask |
| WG      | xxx.xxx.xxx.xxx  
(eg. 192.168.11.1) | Set Gateway |
| WP      | 0~65535 | Set Local IP’s port number |
| WD      | 0 : Static  
1 : DHCP | Set the IP configuration method |
| WM      | 0 : TCP server  
1 : TCP client  
2 : UDP | Set the operation mode |
| WC      | 0 : startup  
1 : any character | TCP client method |
| WB      | XXXX  
eg. [Baudrate]1: 115200, 2: 57600,  
3: 38400, 4: 19200, 5: 9600,  
6: 4800, 7: 2400,8: 1200  
[data byte] 7 : 7bit, 8bit  
[parity] 0 : no parity, 1 : Odd, 2 :Even  
[Flow] 0 : no, 1 : Xon/Xoff, 2 :RTS/CTS | Set the serial baud rate, data, parity and flow control.  
4bytes:[Baud][data byte][parity][flow] |
| WT      | 0 : Disable  
1 : H/W trigger  
2 : S/W trigger | Set the serial command method |
| WE      | xxxxxxx  
(eg. In hex format : 2B 2B 2B) | Set the command mode character |
<table>
<thead>
<tr>
<th>Command</th>
<th>Parameter</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>WX</td>
<td>xxx.xxx.xxx.xxx (eg. 192.168.11.144)</td>
<td>Set server IP address</td>
</tr>
<tr>
<td>WN</td>
<td>0~65535</td>
<td>Set server port number</td>
</tr>
<tr>
<td>WR</td>
<td></td>
<td>Restart</td>
</tr>
<tr>
<td>OC</td>
<td>XX</td>
<td>Set delimiter character in hex</td>
</tr>
<tr>
<td>OS</td>
<td>0~255</td>
<td>Set delimiter size</td>
</tr>
<tr>
<td>OT</td>
<td>0~65535</td>
<td>Set delimiter time</td>
</tr>
<tr>
<td>OI</td>
<td>0~65535</td>
<td>Set Inactivity timer value</td>
</tr>
<tr>
<td>RI</td>
<td></td>
<td>Get Local IP</td>
</tr>
<tr>
<td>RS</td>
<td></td>
<td>Get Subnet mask</td>
</tr>
<tr>
<td>RG</td>
<td></td>
<td>Get Gateway</td>
</tr>
<tr>
<td>RP</td>
<td></td>
<td>Get Local IP's port number</td>
</tr>
<tr>
<td>RD</td>
<td></td>
<td>Get the IP configuration method</td>
</tr>
<tr>
<td>RM</td>
<td></td>
<td>Get the operation mode</td>
</tr>
<tr>
<td>RC</td>
<td></td>
<td>Get the TCP client method</td>
</tr>
<tr>
<td>RB</td>
<td></td>
<td>Get the serial baud rate</td>
</tr>
<tr>
<td>RT</td>
<td></td>
<td>Get the serial command method</td>
</tr>
<tr>
<td>RE</td>
<td></td>
<td>Get the command mode character</td>
</tr>
<tr>
<td>RF</td>
<td></td>
<td>Get the firmware version</td>
</tr>
<tr>
<td>RX</td>
<td></td>
<td>Get the server IP address</td>
</tr>
<tr>
<td>RN</td>
<td></td>
<td>Get the server port number</td>
</tr>
<tr>
<td>QC</td>
<td></td>
<td>Get delimiter character in hex</td>
</tr>
<tr>
<td>QS</td>
<td></td>
<td>Get delimiter size</td>
</tr>
<tr>
<td>QT</td>
<td></td>
<td>Get delimiter time</td>
</tr>
<tr>
<td>QI</td>
<td></td>
<td>Get Inactivity timer value</td>
</tr>
</tbody>
</table>
2.4. Operation Flow

2.5. Factory Default

While the Factory Reset is low and the /Reset is applied, the module is initialized with the factory default value.

<table>
<thead>
<tr>
<th>IP configuration</th>
<th>Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local IP address</td>
<td>192.168.11.2</td>
</tr>
<tr>
<td>Subnet mask</td>
<td>255.255.255.0</td>
</tr>
<tr>
<td>Gateway address</td>
<td>192.168.11.1</td>
</tr>
<tr>
<td>Local port number</td>
<td>5000</td>
</tr>
<tr>
<td>Server IP address</td>
<td>192.168.11.3</td>
</tr>
<tr>
<td>Server port number</td>
<td>5000</td>
</tr>
<tr>
<td>Operation mode</td>
<td>TCP server mode</td>
</tr>
<tr>
<td>Serial port</td>
<td>9600 bps 8-N-1</td>
</tr>
<tr>
<td>Serial command method</td>
<td>H/W trigger</td>
</tr>
</tbody>
</table>
3. Demonstration and Test

In this chapter, three examples are given to show how functions of the EG-SR-7150MJ can be tested. The testing environment is as follows:

**Hardware**
- PC having RS-232 serial port.
- EG-SR-7150MJ & Test board

**Software**
- Windows operating system installed on testing PC.
- EG-SR-7150MJ Configuration tool
- Hyper Terminal Program

**Testing Structure**
- Ethernet cross cable to connect the LAN ports of PC and EG-SR-7150MJ.
- RS-232 cable to connect the COM port of PC (usually COM1 or COM2) and serial port of EG-SR-7150MJ-EVB.

3.1. Case 1: Getting IP address using H/W trigger

**STEP1**: Configure the trigger mode as “H/W trigger” in the Configuration Tool.

**STEP2**: Check the serial port setting such as baud rate of the module.

**STEP3**: Start HyperTerminal program and set the serial port of PC to the setting of module checked in STEP2.

**STEP4**: Pull H/W trigger pin to low to enter the serial command mode.

**STEP5**: Use HyperTerminal program to send “<RI>” (command to request IP address).

**STEP6**: HyperTerminal program displays “<S192.168.11.2>”

(It indicates that the command was successfully executed and IP address is 192.168.11.2)

**STEP7**: Pull H/W trigger pin to high to exit the serial command mode

3.2. Case 2: Changing IP address using H/W trigger

**STEP1**: Configure the trigger mode as “H/W trigger” in the Configuration Tool.

**STEP2**: Check the serial port setting such as baud rate of the module.

**STEP3**: Start HyperTerminal program and set the serial port of PC to the setting of module checked in STEP2.

**STEP4**: Pull H/W trigger pin to low to enter serial command mode.

**STEP5**: Use HyperTerminal program to send “<WI192.168.11.10>”

(command to change the IP address as 192.168.11.10)
STEP6: HyperTerminal program displays "<S>"
   (Indicates the command was successfully executed)

STEP7: Use HyperTerminal program to send "<RI>" (command to request IP address)

STEP8: HyperTerminal program displays "<S192.168.11.10>"
   (Indicates the command was executed successfully and IP address is 192.168.11.10)

STEP9: Pull H/W trigger pin to high to exit serial command mode
☞ All changes are applied after exit the serial command mode.

3.3. Case 3: Changing IP address using S/W trigger

STEP1: Configure the trigger mode as S/W trigger at the Configuration program, and check the three trigger characters. For example, assume the trigger is “25 25 25”

STEP2: Check the serial port setting such as baud rate of the module.

STEP3: Start HyperTerminal program and set the serial port of the PC to the serial setting of the module checked in STEP2.

STEP4: Use HyperTerminal program to send three trigger characters to enter the serial command mode; %%% (in hex :0x25 0x25 0x25) in this case.

STEP5: Use HyperTerminal program to send "<WI192.168.11.10>”
   (command to change the IP address as 192.168.11.10)

STEP6: HyperTerminal program displays "<S>"
   (Indicate the command was executed successfully)

STEP7: Use HyperTerminal program to send "<RI>" (command to request IP address)

STEP8: HyperTerminal program displays "<S192.168.11.10>“
   (Indicate the command was executed successfully and IP address is 192.168.11.10)

STEP9: Use HyperTerminal program to send “<WR>”
   (command to exit serial command mode)
☞ All changes are applied after exiting the serial command mode.
## 4. PIN Assignment and Dimensions

<table>
<thead>
<tr>
<th>Name</th>
<th>Functions</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3V</td>
<td>Power</td>
<td></td>
</tr>
<tr>
<td>/RESET</td>
<td>Low active reset</td>
<td>Input</td>
</tr>
<tr>
<td></td>
<td>Minimum 1.2 usec is required.</td>
<td></td>
</tr>
<tr>
<td>RXD</td>
<td>RS-232 Data Input</td>
<td>Input</td>
</tr>
<tr>
<td>CTS</td>
<td>RS-232 Clear To Send</td>
<td>Input, Optional</td>
</tr>
<tr>
<td>TXD</td>
<td>RS-232 Data Output</td>
<td>Output</td>
</tr>
<tr>
<td>RTS</td>
<td>RS-232 Request To Send</td>
<td>Output, Optional</td>
</tr>
<tr>
<td>Factory Reset</td>
<td>Pull Factory Reset to low and if /RESET is activated, the configuration is changed to factory default.</td>
<td>Input</td>
</tr>
<tr>
<td>H/W Trigger</td>
<td>Pull H/W Trigger to low, enter the serial command mode</td>
<td>Input</td>
</tr>
<tr>
<td>/PSEN</td>
<td>Pull /PSEN to low and if /RESET is activated, the module enter the bootloader for FLIP connection</td>
<td>Input</td>
</tr>
</tbody>
</table>

☞ All signal levels are 3.3V LV TTL.
Ethernet port Pinouts

<table>
<thead>
<tr>
<th>Pin</th>
<th>Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TX+</td>
</tr>
<tr>
<td>2</td>
<td>TX-</td>
</tr>
<tr>
<td>3</td>
<td>RX+</td>
</tr>
<tr>
<td>6</td>
<td>RX-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Dimension(mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>40</td>
</tr>
<tr>
<td>B</td>
<td>62</td>
</tr>
<tr>
<td>C</td>
<td>15</td>
</tr>
<tr>
<td>D</td>
<td>20</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
</tr>
<tr>
<td>F</td>
<td>2</td>
</tr>
</tbody>
</table>
6. ETC

6.1. Firmware Uploading through the FLIP software

The following items are required to get started:

- EG-SR-7150-MJ test board
- UART cross cable
- Program file in HEX file format
- FLIP utility installed on your PC

On a PC, one must have a file in HEX format to program the EG-SR-7150MJ. For example this file could be named “app.hex”.

**Step 1**

Connect the EG-SR-7150MJ test board to a PC with the UART cable supplied.
Important: If you have any program running on your PC with which COM port is used such as “Hyperterminal”, be sure to connect the cable to the COM port not used.

**Step 2**
Power on the test board.
While pressing the /PSEN button, click the /RESET button. Then release the /PSEN button.

**Step 3**
Run the ISP software named FLIP by ATMEL.

**Step 4**
Select the device by pushing the F2 button. Here you must choose AT89C51RC2.

**Step 5**
Set up the communication port by pushing the F3 button. Make sure to select the same port as the one you have plug in the UART cable of the EG-SR-7150MJ test board.

**Step 6**
Now, you should be connected to the board and able to program.
Now you will have to browse your PC to load your file in hex format.

**Step 7**
After programming, check if the BSB, SBV and SSB are set as FF, 00 and FF respectively.

### 6.2. Warranty

WIZnet Co., Ltd offers the following limited warranties applicable only to the original purchaser. This offer is non-transferable.

WIZnet warrants our products and its parts against defects in materials and workmanship under normal use for period of standard ONE(1)YEAR for the EG-SR-7150MJ Module, Evaluation Board and labor warranty after the date of original retail purchase. During this period, WIZnet will repair or replace a defective products or part free of charge.

**Warranty Conditions:**
1. The warranty applies only to products distributed by WIZnet or our official distributors.
2. The warranty applies only to defects in material or workmanship as mentioned above in 6.2 Warranty. The warranty applies only to defects which occur during normal use.
and does not extend to damage to products or parts which results from alternation, repair, modification, faulty installation or service by anyone other than someone authorized by WIZnet Inc.; damage to products or parts caused by accident, abuse, or misuse, poor maintenance, mishandling, misapplication, or used in violation of instructions furnished by us; damage occurring in shipment or any damage caused by an act of God, such as lightening or line surge.

Procedure for Obtaining Warranty Service

1. Contact an authorized distributors or dealer of WIZnet Inc. for obtaining an RMA (Return Merchandise Authorization) request form within the applicable warranty period.
2. Send the products to the distributors or dealers together with the completed RMA request form. All products returned for warranty must be carefully repackaged in the original packing materials.
3. Any service issue, please contact to sales@wiznet.co.kr