The Intelligent Surveillance Solution

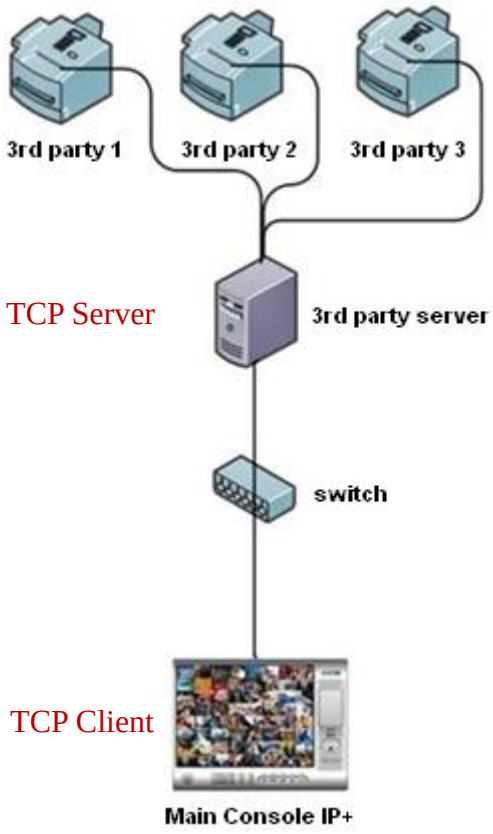
# 3rd Party Integration - Generic TCP/IP Receiver

User Manual

# Table of content

# Introduction

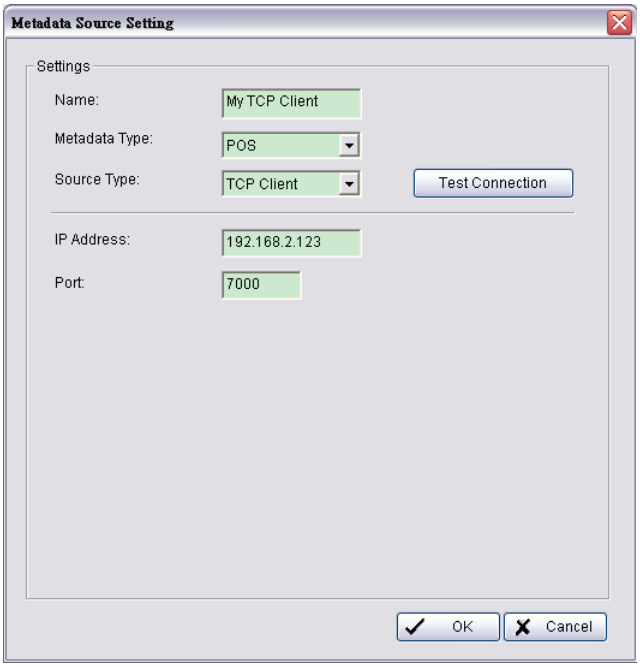
This document describes how to make connection from 3rd party application to recording servers. The server of this 3rd party application will act as TCP Server role and MainConsole will act as TCP Client role. The diagram is as follows:



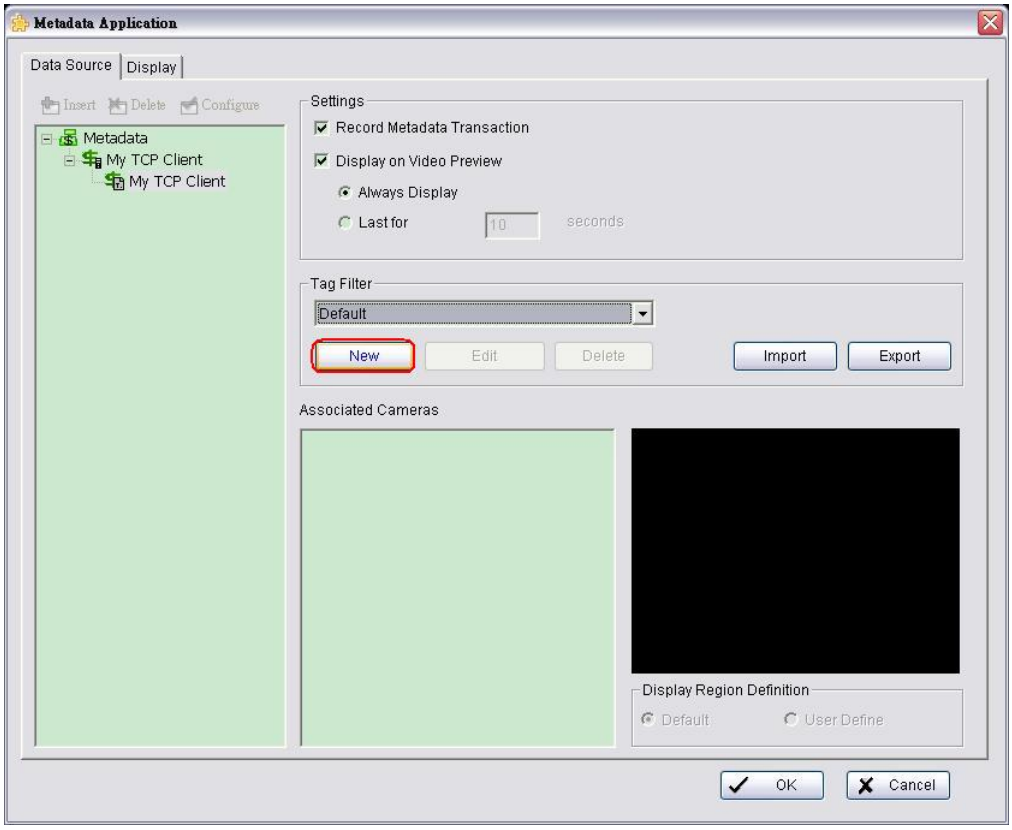

# Integration Steps

Following are the steps to make MainConsole receive data:

1. In the server of 3rd party application, create a TCP Server and listen to a specific port in order to transmit transaction data.
2. Open MainConsole.
3. Create a new Metadata Source. In Metadata Source setting, use POS as Metadata Type, TCP Client as source type and fill the IP address/port which the TCP Server listen to. If the TCP service build correctly, "Test Connection" will return success.

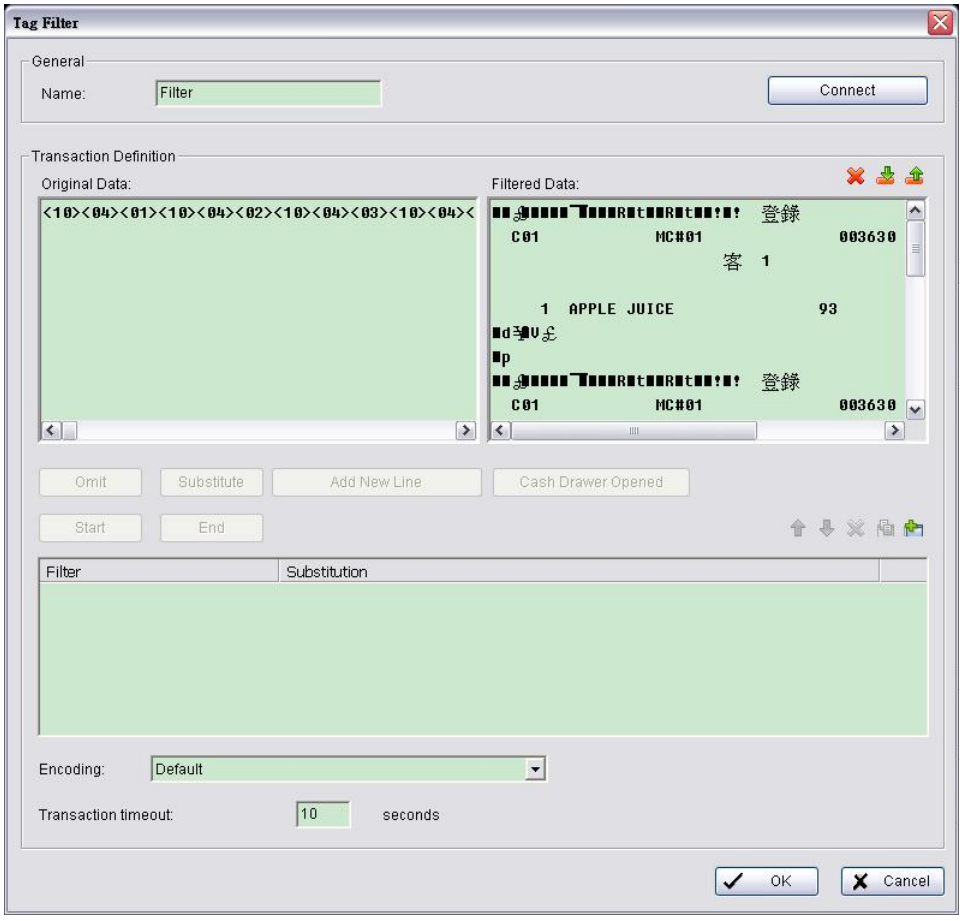



4. TCP Server sends metadata, then MainConsole receives metadata. Using tag filter to configure page to check sending data. Press "New" to create a new filter.




After press "Connect", MainConsole will connect to TCP Server,

receive data from TCP Server.



# TCP Server Sample

```
class CSocketHelper
{
    // Use specific port to allow tcp client connect
    bool  Listen(int port, int maxUser);

    // Send data to a connected socket
    void  Send(const char *buf, int len);
    …..
    SOCKET    _socket;
}
bool CSocketHelper::Listen(int port, int maxUser)
{
  int nRet;
  SOCKET listeningSocket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

   int err = WSAGetLastError();
  if (listeningSocket == INVALID_SOCKET) {
      return false;
  }

  // Use SOCKADDR_IN to fill in address information
  SOCKADDR_IN saServer;
  saServer.sin_family = AF_INET;
  saServer.sin_addr.s_addr = INADDR_ANY;
  saServer.sin_port = htons(port);

  // Bind the socket to our local server address
  nRet = bind(listeningSocket, (LPSOCKADDR)&saServer, sizeof(struct
sockaddr));
  if (nRet == SOCKET_ERROR) {
       closesocket(listeningSocket);
      return false;
  }

  // Make the socket listen
  nRet = listen(listeningSocket, maxUser);
  if (nRet == SOCKET_ERROR) {
       closesocket(listeningSocket);
      return false;
  }

   if (_socket)
   {
      ASSERT(0);
     closesocket(_socket);
      _socket = 0;
   }

  _socket = listeningSocket;
  _bAutoClose = true;
  return true;
}

void CSocketHelper::Send (const char *buf, int len)
```

```
{
  while (len > 0)
  {
      int s = send(_socket, buf, len, 0);

      if (s <= 0)
          throw WSAGetLastError();

      buf += s;
      len -= s;
  }
}
```

---

```
// This class is to monitor which TCP Client connect in and send metadata when
there is new data
class CMetadataTCPResponder : public Thread, public CSocketHelper
{
public:
    virtual void Do();
    ….

    Std::vector< SOCKET> m_ConnectedSockets;
};
// This class maintain the TCP Service, to start or stop
class CTCPSessionMgr  : public Thread
{
    friend class CMetadataTCPResponder;
    BOOL StartService();
    BOOL StopService()=0;
    BOOL ResetService()=0;
……
  CSocketHelper  _service;
}

void CMetadataTCPResponder::Do()
{
  //someone connected to this TCP server,
    //send data when there is data. Need to remember who connect in
    SOCKET socket;
    Delegate(socket);
  m_ConnectedSockets.pushback(socket);

    while (…) // If there is new data, send to all connected clients
    {
        try
        {
          ……
          Send(socket , strMetadata, strMetadata.GetLength());
        }
        catch (...)
        {
            // can’t send data the current socket, the socket may not exist
            // remove this socket from vector
        }
     }
}
```

```
BOOL CTCPSessionMgr:: StartService ()
{
  // start to listen someone conntect
  if (!_service.Listen(_port, 10))
      return FALSE;

  return CNuThread::Create();
}
```