ScriptRunner

1. What is ScriptRunner gadget?
   ScriptRunner gadget is a gadget that receives data from multiple sources, processes it and can send the results to multiple output pins (on the right hand side of the gadget). The data processing is done according to a script loaded onto the gadget. This gadget is unique in the sense that changes can be made in the script while it runs, and will apply once it is loaded onto the gadget; there is no need to restart the program or even to restart the running graph. Programmers note: ScriptRunner gadget and the Script use the same address space, allowing easy access to the same data from both codes.

Figure 1: ScriptRunner Gadget

2. Maximum number of ScriptRunner gadgets
   Upon Stream Handler's installation, there exists a limitation of a total of 10 Script Runner gadgets in all open graphs. If the user creates additional Script Runner gadgets, he is requested to save the graph\s, close it and reload it. The maximum number of Script Runners will be adjusted so that the graph will function properly upon reloading.

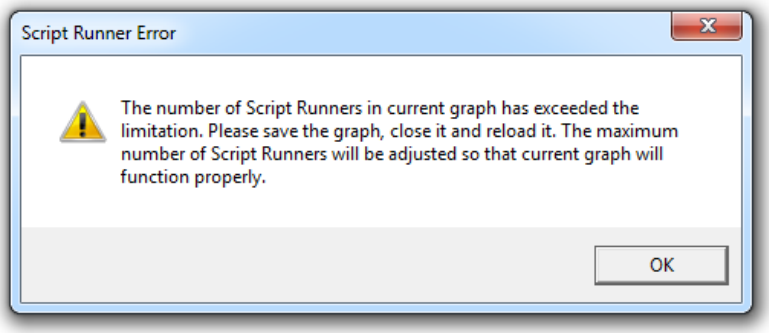


Figure 2: Number of Script Runner Exceeded the Limit

3. How to use ScriptRunner gadget?

3.1 Script Runner gadget's input pins can be connected to the following gadgets:

3.1.1 Text gadget of any type

3.1.2 TVObjects gadget

3.1.3 Sink gadget (that can be connected to any of the above gadgets)

3.2 How to load a script?

3.2.1 Double left click / right click and 'Setup' selection on the gadget creates the following dialog box:

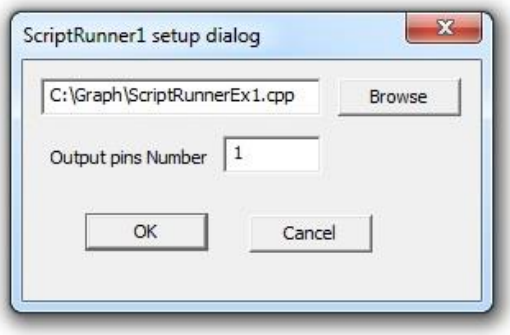


Figure 3: ScriptRunner Gadget Setup Dialog

The user can either enter the path to the script manually (i.e. C:\Graph\NewScript.cpp) or select it through the file browser.

3.2.2 The content of any text data packet received in control pin is considered as the path to the script and ScriptRunner gadget attempts to load it.
The script must be of one of the following types: cpp / c / ch / txt

3.3 How to change the number of outputs pins?
Double left click / right click and 'Setup' selection on creates the dialog box shown in Figure 1. Changing the number to the right of the 'Output Pins Number' text and pressing 'OK' will cause the gadget to realign with more / less output pins according to the input.

3.4 Interaction with TVObjects gadget
As mentioned above, ScriptRunner gadget can connect and receive data from any text capture gadget. There is, however, one gadget whose text data receive special care – TVObjects gadget. TVObjects gadget determines which objects will be searched for in the video frames and sends the data regarding the detected objects along with the video frame to its output pin. If TVObjects gadget is connected to ScriptRunner gadget, the data regarding the detected objects and the video frame is stored in a predetermined data structure which is sent to the script (whereas text data from any other gadget is sent to the script in a much simpler predetermined text data structure). This enables easy access to the data regarding the detected objects and the video frame from the script whereas text data from any other gadget must be parsed in the script in order to gain access to specific parts of the data.

3.5 How to receive data from multiple gadgets (Sink gadget)?
ScrintRunner gadget has only one input pin. In order to receive data from multiple sources, Sink gadget must be added. Sink gadget has up to 999 input pins, adds the input pin index to the data it receives and sends it to its output pin. ScriptRunner gadget can, according to the script, process data from each pin differently. Programmers note: any other user-created-gadget that performs the same tasks as the Sink gadget (i.e. sets the label as "Sink", creates new quantity frame, sets its value as the pin's index number and adds the quantity frame to the container) can be used instead of Sink gadget.

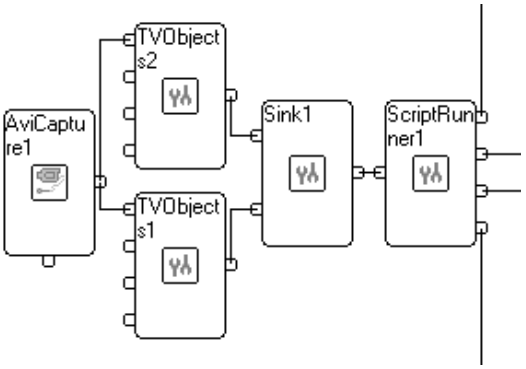


Figure 4: Receiving Information from Multiple Sources

4. How to write a Script?

Scripts are written in c programming language which means that some data types used in c++ (like CString) are not supported.

4.1 Basic script

The following script corresponds to graphs in which text gadget (in this case Generic Capture gadget) is connected to ScriptRunner gadget.

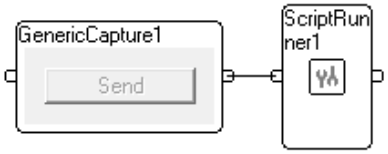


Figure 5: ScriptRunner and GenericCapture Gadget

Scripts are generally composed of 4 parts:

4.1.1 Administrative:

Contains global variables, data structures declarations, global definitions and included files. To enable feedback from script to ScriptRunner gadget, ScriptRunner index and TxtOut_toC function must be declared in this part.

4.1.1.1 Example

```
int __SCRIPTRUNNER_INDEX__; // Declares the ScriptRunner index to distinguish it
                            // from other ScriptRunner gadgets in the graph
extern int TxtOut_toC (int iOutPinNumber ,char * pOut,
int __SCRIPTRUNNER_INDEX__ ); // Enables the sending of the string pOut from
                              // script to iOutPinNumber output pin in
                              // ScriptRunner gadget.
// If the string pOut contains the characters “%T”, these characters are replaced
// by the Total Processing Time (time duration between capturing the frame and
// finishing executing the script).
// “%S” is replaced by Script Processing Time (time duration between data
// structure ready to be sent to script (see 3.2.2) and finishing executing the
// script).
// “%B” is replaced by SHStudio Processing Time (time duration between capturing
// the frame and converting it to data structure) and “%C” is replaced by current
// time. Naturally, if none of those exists, the pOut reminds as it is.
class TxtToScript // Defines the predetermined text data structure
```

```
{
public:
  double GraphTime; // Graph time [ms] in which the text was sent to script
  char* pTxtOut;    // Pointer to the text
};
char Msg[300]; // Declares Msg variable used to send massages to
               // ScriptRunner gadget via TxtOut_toC function
```

4.1.2 OnInit_Ch (char* pIn, char* pOut)

This function defines the reaction to changes in the graph's execution status or to changes in ScriptRunner gadget. The possible contents of pIn are:

4.1.2.1 'INIT'. This input is sent to the script when a script is loaded (through the dialog box or when graph is loaded - in this case it can be used to send initializing messages to other gadgets).

4.1.2.2 'START'. This input is sent to the script when start button is pressed only if the ScriptRunner gadget is connected directly or indirectly to capture gadget.

4.1.2.3 'STOP'. This input is sent to the script when stop button is pressed, when a running graph is closed (since it stops and then closes) or after initialization (if start button was not pressed) only if ScriptRunner gadget is connected directly or indirectly to capture gadget.

The user can define different reaction to each input.

4.1.2.4 Example

```
int OnInit_Ch(char* pIn, char* pOut)
{
  if (strstr( pIn , "INIT")) // OnInit_Ch function with input "INIT"
  {
   sprintf( Msg , "INIT command" ) ;
    return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
  }
  if (strstr( pIn , "START")) // OnInit_Ch function with input "START"
  {
    sprintf( Msg , "START command" ) ;
    return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
  }
  if (strstr( pIn , "STOP")) // OnInit_Ch function with input "STOP"
  {
     sprintf( Msg , "STOP command" ) ;
     return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
  }
  else
    return 0;
}
```

4.1.3 OnTxtStructInput_Ch (int iInpIndex , TxtToScript* pTxtOut)

This function defines the data processing based on the text data and the input pin index from which the data was received (see: 2.5). If text gadget is connected directly to ScriptRunner gadget (not through Sink gadget) the input pin index is 0.

4.1.3.1 Example

```
int OnTxtStructInput_Ch (int iInpIndex , TxtToScript* pTxtOut )
{
  sprintf( Msg , "INPUT processing " ) ;
```

```
  if (strstr (pTxtOut->pTxtOut, "test string")) // If text input contains "test
                                                // string", send the text "test
                                                // string found! " to output pin
                                                // in ScriptRunner gadget. Else
                                                // send the text "INPUT
                                                // processing"
    sprintf( Msg , "test string found! " ) ;
  return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
}
```

4.1.4 OnExit_Ch (char* pIn, char* pOut)

This function defines the reaction to graph closure or to the reloading of a script.

4.1.4.1 Example

```
int OnExit_Ch (char* pIn, char* pOut)
{
  if ( strstr( pIn , "EXIT" ) ) // OnExit_Ch function with input "EXIT"
  {
       sprintf( Msg , "EXIT command" ) ;
       return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
  }
}
```

This basic script can be found loaded to ScriptRunner gadget in ScriptRunnerEx1.tvg.

Note: in order to run this and the other example graphs, they must be placed along with their scripts in C:\Graph.

4.2 TVObjects script

The following script corresponds to graphs in which TVObjects gadget is connected to ScriptRunner gadget.

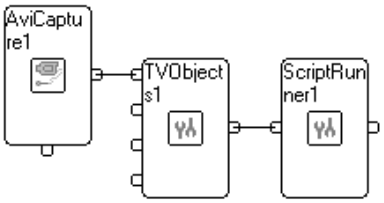


Figure 6: ScriptRunner and TVObjects Gadget

As mentioned before, If TVObjects gadget is connected to ScriptRunner gadget, the data regarding the detected objects and the video frame is stored in a predetermined data structure which is sent to the script whereas text data from any other gadget is sent to the script in a much simpler predetermined text data structure (see 3.1.1.1).

4.2.1 Administrative:

```
int __SCRIPTRUNNER_INDEX__; // Declares the ScriptRunner index to distinguish it
                            // from other ScriptRunner gadgets in the graph
extern int TxtOut_toC(int iOutPinNumber, char * pOut, int __SCRIPTRUNNER_INDEX__);
                           // Enables the sending of the string pOut from script to
                           // iOutPinNumber output pin in ScriptRunner gadget.
extern int FrameOut_toC(int iOutPinIndex, void * pFrame, int__SCRIPTRUNNER_INDEX__);
                    // Enables the sending of the video frame pFrame from
                    // script to iOutPinNumber output pin in ScriptRunner gadget.
typedef unsigned long DWORD; // Define data type DWORD used in TVObject data
                             // structure
class TxtToScript // Defines the predetermined text data structure.
```

```
{
public:
   double GraphTime; // Graph time [ms] in which the text was sent to script
   char* pTxtOut;    // Pointer to the text
};
typedef struct PointToScript // Define points data structure used
                             // in the TVObjects data structure.
{
   double x , y;
   int IsROI; // Determines if the point defines the Region
              // Of Interest or a detected point
}
PointToScript;
class NodeToScript
// Defines the predetermined TVObjects data structure. Each such data structure
// is part of a linked list (contains a pointer to the previous data structure).
// Every data structure can contain data about one the following data frames:
// 1. FigureFrame - data regarding all the figures found in the frame.
// 2. TextFrame - data regarding all the texts found in the frame.
// 3. InfoFrame – data regarding the entire linked list
{
public:
   int m_iType; // The type of current data structure. Possible values: 0 = figure
                // frame, 1 = text frame, 2 = info frame
   NodeToScript* m_pPrev; // Pointer to the previous data structure in current
                          // linked list.
   char* m_pName; // Pointer to the label of the figure or text frame
   PointToScript* m_pPoints; // Pointer to the array containing the coordinates
                             // of the detected objects
   int m_nPoints;  // Number of detected points
   char* m_pValue; // String of detected text
   int m_DoDelete; // Determines whether data structure should be deleted after the
                   // data was acquired by the script
   void *pFrame;   // Pointer to the video frame after TVObjects gadget's analysis
   void *pOriginalFrame; // Pointer to the original video frame before TVObjects
                         // gadget's analysis
   DWORD CameraTime; // Frame creation in the camera time stamp determined by camera
   DWORD CameraFrameNumber ; // Frame number determined by camera
   DWORD CameraOutP; // Camera input
   DWORD CameraInP; // Camera output
   DWORD CameraX; // X coordinate determined by camera
   DWORD CameraY; // Y coordinate determined by camera
   int m_iId; // Frame ID
   double m_dFrameTime; // Data frame creation in graph time stamp
   double m_dBeforeScriptTime; // Before transmitting data structure to script time
                               // stamp
   int m_iInputIndex; // Input pin index
   int m_iNfigs; // Number of figure frames in data structure
   int m_iNTexts; // Number of text frames in data structure
   NodeToScript** pFigs; // Array of all the figure pointers in data structure
   NodeToScript** pTxts; // Array of all the text pointers in data structure
};
char Msg[300]; // Declares Msg variable used to send massages to ScriptRunner gadget
               // via TxtOut_toC function
```

4.2.2 OnDataStructInput_Ch (int iInpIndex , NodeToScript * pIn)

While OnInit_Ch (char * pIn, char * pOut) and OnExit_Ch (char * pIn, char * pOut) functions may remain the same, the OnTxtStructInput_Ch (int iInpIndex , TxtToScript* pTxtOut) must change in

order to be able to receive the TVObject data structure. Note that this function is called for every data frame even if there was no detection in TVObjects.

```
int OnDataStructInput_Ch (int iInpIndex , NodeToScript * pIn )
// Defines the data processing based on the TVObjects data structure and the input
// pin index from which the data was received. If TVObjects gadget is connected
// directly to ScriptRunner gadget (not through Sink gadget) the input pin index is
// 0.
{
  sprintf( Msg, "INPUT processing " ) ;
  return TxtOut_toC(3, Msg, __SCRIPTRUNNER_INDEX__ );
}
```

4.2.2.1 Example

The following code demonstrates how to access information (in this case, the coordinates of the detected points) in TVObject data structure from within the script and how to send specific frames to the output pin of ScriptRunner gadget:

```
int OnDataStructInput_Ch (int iInpIndex , NodeToScript * pIn )
{
  int i = 0 ;
  NodeToScript* p = pIn ; // Casting the input to the data structure
  if (p->m_iNfigs > 0) // If there exists figure frames
  {
    for (i = 0; i < p->m_iNfigs; i++) // Iterates over all the figure frames
    {
      NodeToScript* pFigs = p->pFigs[i] ;
      int j = 0 ;
      for (j = 0; j < pFigs->m_nPoints; j++) // Iterates over all the points in
                                             // current figure frame
      {
        if (pFigs->m_pPoints[j].IsROI != 1) // If the points are the detected
                                            // points, not the points that define
                                            // the TVObject's Region Of Interest
        {
          sprintf ( Msg , " Detected Point Coordinates: X = %6.2lf, Y =
            %6.2lf", pFigs->m_pPoints[j].x, pFigs->m_pPoints[j].y ) ;
            // Create text message with points coordinates
          if ( pFigs->m_pPoints[j].x > 350 ) // If x coordinate is greater than 350
              FrameOut_toC(1, pFigs->pFrame, __SCRIPTRUNNER_INDEX__ ); // Send
                                        // video frame to the first output pin
          TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ ); // Send text message to
                                                         // the first output pin

        }
      }
    }
  }
  else // No figure frames
  {
    sprintf ( Msg , "No detected Points!"); // Create text message
    TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ ); // Send text message to the
                                                   // first output pin

  }
  return 1;
}
```

This TVObjects script can be found loaded to ScriptRunner gadget in ScriptRunnerEx2.tvg. Note: in order to run this and the other example graphs, they must be placed along with their scripts in C:\Graph.

4.3 Sink script:

The following script corresponds to graphs in which TVObjectGadget/s is/are connected to Sink gadget which is connected to ScriptRunner gadget.

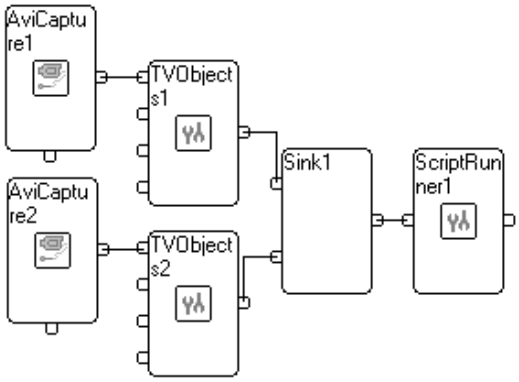


Figure 7: ScriptRunner and Sink Gadget

As mentioned before, ScriptRunner gadget can receive and process data from multiple sources using Sink gadget and an appropriate script. Notice that if more than one Sink gadgets are connected in a row, the input pin indexes which are sent to the ScriptRunner gadget are of the Sink gadget directly connected to the ScriptRunner gadget.

The following example demonstrates the usage of a switch on the input pin index and the appliance of different processing to data from different inputs, in function OnDataStructInput_Ch (int iInpIndex , NodeToScript * pIn).

4.3.1 Example

```
int OnDataStructInput_Ch (int iInpIndex , NodeToScript * pIn )
{
  switch ( iInpIndex )   // switch according to the input pin index
  {
    case 0 :
    {
       sprintf( Msg , "INPUT processing - data from input 0 " ) ;
       return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
    }
   case 1 :
   {
      sprintf( Msg , "INPUT processing - data from input 1 " ) ;
      return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
    }
  }
       }
```

This Sink script can be found loaded to ScriptRunner gadget in ScriptRunnerEx3.tvg.

Note: in order to run this and the other example graphs, they must be placed along with their scripts in C:\Graph.

4.4 Complete Script #1

The following script corresponds to graphs in which 2 TVObject gadgets and 1 text gadget (in this case GenericCapture gadget) are connected to Sink gadget which is connected to ScriptRunner gadget.

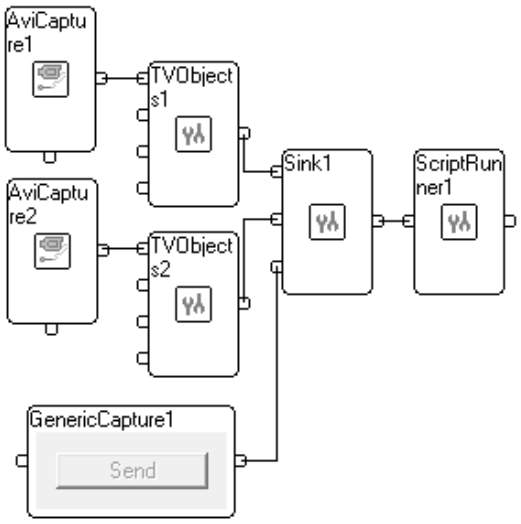


Figure 8: Graph for Complete Script #1

This script is a merge of some of the scripts presented earlier and therefore will contain no remarks or explanations.

```
int __SCRIPTRUNNER_INDEX__;
extern int TxtOut_toC (int iOutPinNumber ,char * pOut, int __SCRIPTRUNNER_INDEX__);
extern int FrameOut_toC (int iOutPinIndex , void * pFrame, int
__SCRIPTRUNNER_INDEX__);
typedef unsigned long DWORD;
class TxtToScript
{
public:
  double GraphTime;
  char* pTxtOut;
};
typedef struct PointToScript
{
  double x , y;
  int IsROI;
}
PointToScript;
class NodeToScript
{
public:
   int m_iType;
   NodeToScript * m_pPrev;
   char* m_pName;
```

```
   PointToScript * m_pPoints;
   int m_nPoints;
   char* m_pValue;
   int m_DoDelete;
   void *pFrame;
   void *pOriginalFrame;
   DWORD CameraTime;
   DWORD CameraFrameNumber ;
   DWORD CameraOutP;
   DWORD CameraInP;
   DWORD CameraX;
   DWORD CameraY;
   int m_iId;
   double m_dFrameTime;
   double m_dBeforeScriptTime;
   int m_iInputIndex;
   int m_iNfigs;
   int m_iNTexts;
   NodeToScript** pFigs;
   NodeToScript** pTxts;
};
char Msg[300];
int OnInit_Ch(char* pIn, char* pOut)
{
  if (strstr( pIn , "INIT"))
  {
   sprintf( Msg , "INIT command" ) ;
    return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
 }
  if (strstr( pIn , "START"))
  {
    sprintf( Msg , "START command" ) ;
    return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
  }
  if (strstr( pIn , "STOP"))
  {
     sprintf( Msg , "STOP command" ) ;
     return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
  }
  else
    return 0;
}
int OnDataStructInput_Ch (int iInpIndex , NodeToScript * pIn )
{
  switch ( iInpIndex )
  {
  case 0 :
  {
      int i = 0 ;
      NodeToScript * p = pIn ;
      if (p->m_iNfigs > 0)
      {
        for ( i = 0  ; i < p->m_iNfigs; i++ )
        {
          NodeToScript * pFigs = p->pFigs[i] ;
```

```
            int j = 0 ;
            for ( j = 0  ; j < pFigs->m_nPoints ; j++ )
            {
              if (pFigs->m_pPoints[j].IsROI != 1)
              {
                sprintf( Msg , "INPUT processing - data from input 0 " ) ;
                 TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
                 sprintf ( Msg , " Detected Point Coordinates: X = %6.2lf, Y =
                 %6.2lf", pFigs->m_pPoints[j].x, pFigs->m_pPoints[j].y ) ;
                 if (pFigs->m_pPoints[j].x > 350)
                   FrameOut_toC( 1 , pFigs->pFrame, __SCRIPTRUNNER_INDEX__ );
                TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
              }
            }
          }
        }
        else
        {
          sprintf ( Msg , "No detected Points!") ;
          TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
        }
        return 1;
    }
    case 1 :
      {
        sprintf( Msg , "INPUT processing - data from input 1 " ) ;
        return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
      }
    }
}
int OnTxtStructInput_Ch (int iInpIndex , TxtToScript* pTxtOut )
{
    switch ( iInpIndex )
    {
    case 2 :
      {
         sprintf( Msg , "INPUT processing – data from input 3 " ) ;
         TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
         sprintf( Msg , "The text input is %s ", pTxtOut->pTxtOut) ;
         return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
      }
    }
}
int OnExit_Ch (char * pIn, char * pOut)
{
    if ( strstr( pIn , "EXIT" ) )
    {
      sprintf( Msg , "EXIT command" ) ;
      return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
    }
}
```

This Complete script #1 can be found loaded to ScriptRunner gadget in ScriptRunnerEx4.tvg. Note: in order to run this and the other example graphs, they must be placed along with their scripts in C:\Graph.

4.5 Complete Script #2

The following script corresponds to graphs in which TVObject gadgets is connected to ScriptRunner gadget and demonstrates accessing ocr information and csv file handling:

```
#include <stdio.h>
int __SCRIPTRUNNER_INDEX__;
extern int TxtOut_toC (int iOutPinNumber ,char * pOut, int __SCRIPTRUNNER_INDEX__);
extern int FrameOut_toC (int iOutPinIndex , void * pFrame, int
__SCRIPTRUNNER_INDEX__);
typedef unsigned long DWORD;
class TxtToScript
{
public:
  double GraphTime;
  char* pTxtOut;
};
typedef struct PointToScript
{
  double x , y;
  int IsROI;
}
PointToScript;
class NodeToScript
{
public:
   int m_iType;
   NodeToScript * m_pPrev;
   char* m_pName;
   PointToScript * m_pPoints;
   int m_nPoints;
   char* m_pValue;
   int m_DoDelete;
   void *pFrame;
   void *pOriginalFrame;
   DWORD CameraTime;
   DWORD CameraFrameNumber ;
   DWORD CameraOutP;
   DWORD CameraInP;
   DWORD CameraX;
   DWORD CameraY;
   int m_iId;
   double m_dFrameTime;
   double m_dBeforeScriptTime;
   int m_iInputIndex;
   int m_iNfigs;
   int m_iNTexts;
   NodeToScript** pFigs;
   NodeToScript** pTxts;
};
char Msg[300];
char FileVersion[300];
FILE * fw = NULL ;
char * pCSVNameProto = "c:\\Graph\\ScriptRunnerEx5_%d.csv" ;
int OnInit_Ch(char * pIn, char * pOut)
{
```

```
  if ( strstr(pIn , "INIT") )
  {
    if (fw != NULL)
    {
      fclose (fw) ;
      sprintf( Msg , "File %s is closed" , FileVersion ) ;
      TxtOut_toC( 0 , Msg , __SCRIPTRUNNER_INDEX__ );
      fw = NULL ;
    }
    int version = 1;
    do
    {
      sprintf(FileVersion, pCSVNameProto , version);
      fw = fopen( FileVersion , "r" ) ;
      if ( fw )
      {
        fclose (fw);
        version ++ ;
      }
    } while ( fw ) ;
    fw = fopen( FileVersion , "ab" ) ;
    sprintf( Msg , "File %s is %s opened" , FileVersion , (fw) ? "" :
       "not" ) ;
    TxtOut_toC( 0 , Msg , __SCRIPTRUNNER_INDEX__ );
    sprintf( Msg , "INIT command" ) ;
    return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );

  }
  if (strstr(pIn , "START") )
  {
    sprintf( Msg , "START command") ;
    return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
  }
  if ( strstr(pIn , "STOP") )
  {
    sprintf( Msg , "STOP command") ;
    return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
   }
   else
     return 0;
}
int OnDataStructInput_Ch (int iInpIndex , NodeToScript * pIn )
{
  int i = 0 ;
  NodeToScript * p = pIn ;
  if (p->m_iNTexts > 0)
  {
    for ( i = 0  ; i < p->m_iNTexts ; i++ )
    {
      NodeToScript * pTxts = p->pTxts[i] ;
      char* Value = pTxts->m_pValue;
      sprintf ( Msg, "Detected Text: %s ", Value ) ;
      TxtOut_toC( 0, Msg, __SCRIPTRUNNER_INDEX__ ) ;
      fprintf( fw, "Detected Text: %s,\n ", Value ) ;
    }
  }
  else
```

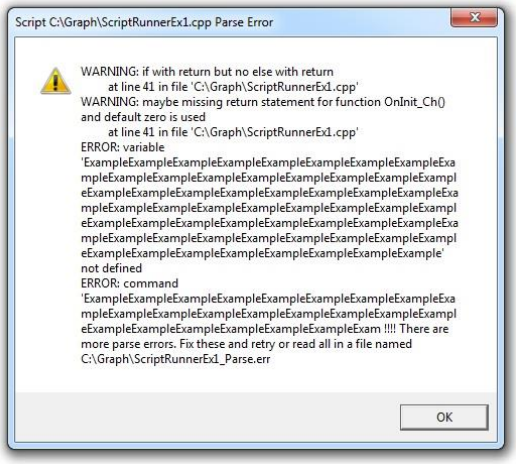
```
  {
    sprintf ( Msg , "No detected Text!") ;
    TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
    fprintf( fw, "No detected Text!,\n" ) ;
  }
  return 1;
}
int OnExit_Ch (char * pIn, char * pOut)
{
  if ( strstr(pIn , "EXIT") )
  {
    if ( fw != NULL )
      fclose( fw ) ;
    fw = NULL ;
    sprintf( Msg , "EXIT command" ) ;
    return TxtOut_toC( 0 , Msg, __SCRIPTRUNNER_INDEX__ );
  }
  else
    return 0 ;
}
```

This Complete script #2 can be found loaded to ScriptRunner gadget in ScriptRunnerEx5.tvg.

Note: in order to run this and the other example graphs, they must be placed along with their scripts in C:\Graph.

5. Errors

In general, error messages are written to a pop up window and to an error file (err). Both the headline of the window and the name of the file are composed of the error type and the location of its occurrence (script / gadget). The error file is located in the same folder as the script which is loaded into the gadget. The error message in the pop up window can contain up to 1000 characters of the code that caused the error and a short description of the error. If the error message contains more than 1000 characters, it will also contain a reference to the error file that contains the entire error message. For example:



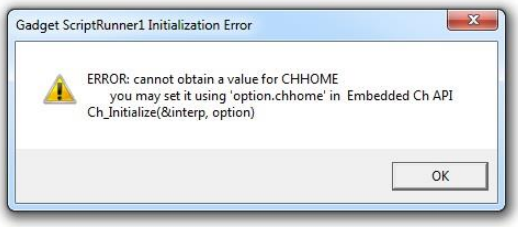
There are several kinds of errors:

5.1 Initialization Error

Each ScriptRunner gadget must be initialized right after its creation (it is created by loading a graph or by dragging a new gadget from the gadgets tree to the left of the graph). This type of error occurs

during this initialization. Since this error is gadget (and not script) related, both the headline of the pop up window and the name of the error file will be composed from the Initialization error type and the ScriptRunner gadget's name.

Example:



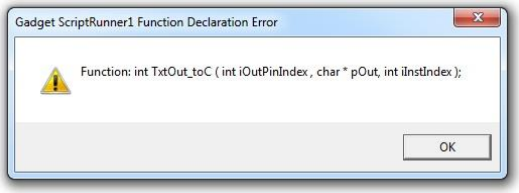
In this example, the error file name will be: ScriptRunner1_Initializtion.err

Solution: seek technical support from FileX ltd.

5.2 Function Declaration Error

ScriptRunner gadget sends and receives information from the Script through a set of predefined, hardcoded (cannot be changed or defined in the script) functions. These functions must be declared after the gadget is initialized. Failure in these declarations causes such errors.

Example:

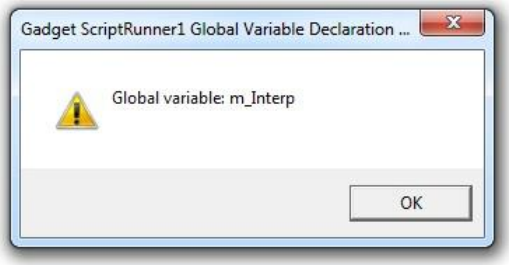


Solution: seek technical support from FileX ltd.

5.3 Global Variable Declaration Error

Much in the same manner, the script runner itself must be accessible from the Script as well as from the ScriptRunnerGadget and therefore must be declared. Failure in this declaration causes such error.

Example:

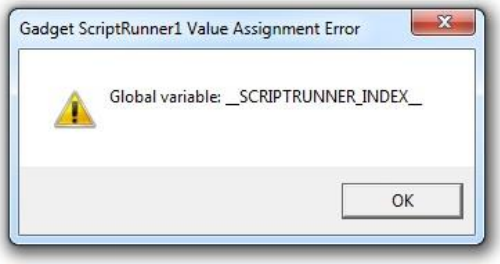


Solution: seek technical support from FileX ltd.

5.4 Value Assignment Error

A graph can contain many ScriptRunner gadgets. The script must be informed about the index of the ScriptRunner gadget to which it is loaded. Failure in relaying this information causes such error.

Example:

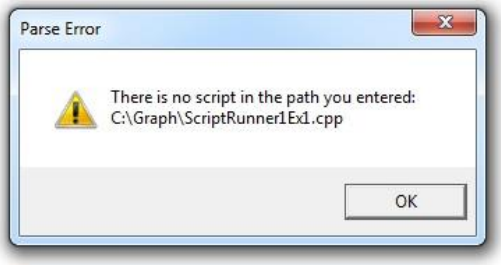


Solution: seek technical support from FileX ltd.

5.5 Parse Error

These errors occur if the script doesn't exist or if the syntax of the script is incorrect and can't be executed.

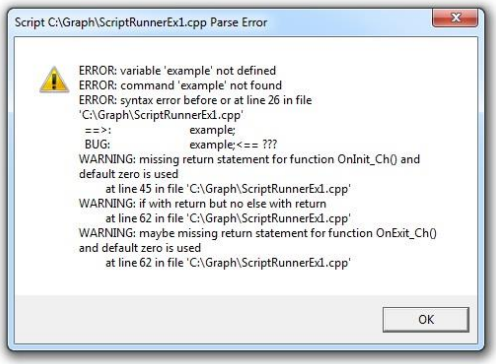
5.5.1 'No such file' Error



Occurs when the user selects a path to a file that doesn't exists.

Solution: the user should choose another file, one that exists, and press 'ok'.

5.5.2 Parse Error



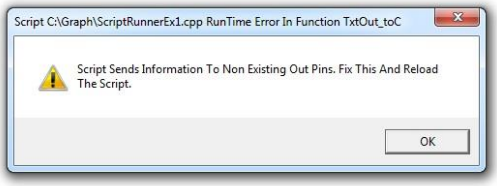
Occurs when the syntax of the script is incorrect and can't be executed.

Solution: fix the programming error and then press 'ok'.

5.6 RunTime Error

Occurs when the script encounters an exception while executing, though the syntax of the script is correct.

5.6.1 'No output pin' Error



Occurs when TxtOut_toC (3 , Msg, __SCRIPTRUNNER_INDEX__) function is called in the script with reference to output index that doesn't exists (e.g. output index = 3 while ScriptRunner gadget has only 1 output pin).

Solution: either change the number of output pins to fit or change the script so it will only send data to existing output pins and press 'ok'.

5.6.2 Runtime Error

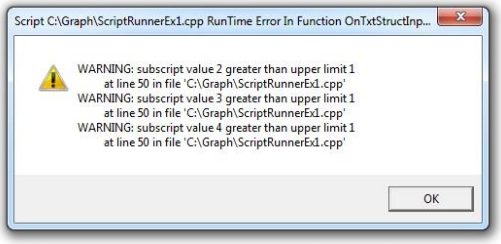
These errors may occur due to numerous reasons.

Example:

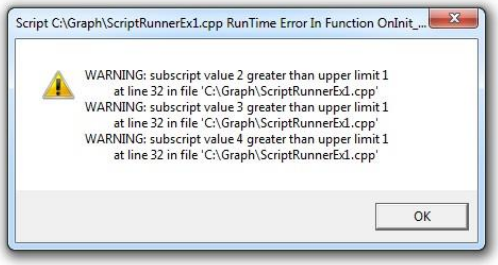
The following code tries to set values in array indexes there were not allocated thus causing a runtime error.

```
int i = 0 ;
int array [2] ;
for ( i = 0 ; i < 5 ; i++)
{
  array[i] = i ;
}
```

The resulting error:



The error message informs the user in which function within the script, the runtime error had occurred. Copying the same code fragment, to OnDataStructInput_Ch results in the following error:



RunTime Error In OnInit_Ch Function must not be confused with Gadget Initialization error (see 4.1).

Solution: fix the programming error and then press 'ok'. Since it is not possible to debug the

script (i.e. observe the variables assume values step by step), it is advisable to use the TxtOut_toC function and create printouts in different parts of the script.