IBM Sterling Gentran:Server for Windows

Script Language Reference Guide

Version 5.3.1

IBM Sterling Gentran:Server for Windows

Script Language Reference Guide

Version 5.3.1

Note

Before using this information and the product it supports, read the information in "Notices" on page 69.

This edition applies to the 5.3.1 version of IBM Sterling Gentran:Server for Microsoft Windows and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. About Sterling Gentran:Server Script Language

Command Format

Commands are shown using the following format:

```
CommandName(string Required_Parameter, [integer Optional_Parameter])
```

Parameters are enclosed in parentheses ( ) and separated by commas (,). Parameter names are case sensitive. If a command does not have any parameters, it is shown with empty parentheses.

The parameters of each command are shown with the data type of the parameters. In the example above, `Required_Parameter` is defined as a string parameter and would be used to hold one or more printable characters. `Optional_Parameter` is defined as an integer parameter and would be used to hold whole numbers that do not have decimal fractions.

Parameters

Commands can have two types of parameters:

- optional parameters—shown inside square brackets ([ ])
- required parameters—shown without square brackets

All parameters in the script commands must be declared before you can use them in the script. A declaration consists of a data type and the parameter that has that type.

Examples

```
string StopRcv;
integer TimeOut;
array line[10];
```

Data types

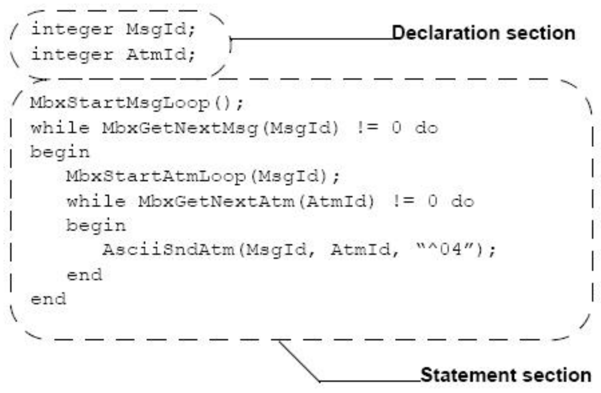
Parameters can have the following data types:

- Integer—a whole number without a decimal component
- Real—a number that may have a decimal component
- String—one or more printable characters
- Datetime—a date or time
- Array—a table of multiple occurrences of a single data type

Script Format

Scripts are divided into two sections: a declaration section used to define the parameters used in the script, and a statements section which holds the actual script commands.

Example



```
integer MsgId;
integer AtmId;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      AsciiSndAtm(MsgId, AtmId, "^04");
   end
end
```

Keywords

A keyword is a special command used in the script to control the flow of the script. Keywords are used in conditional logic to test for matches to defined values.

Example

```
if RcvResult = 1 then
  LogMessage("receive no data condition occurred");
```

In this example a log message is generated when the value of `RcvResult` is equal to the number 1. The `if...then` keywords control the script such that if the value of `RcvResult` does not equal 1, no log message is generated.

The following keywords are available for use in the communications scripts:
- if...then...else
- while...do
- begin...end
- continue...break

Line terminators

Declarations and statements are terminated with a semicolon (;).

Exception: Keyword statements (like `if...then` or `while...do`) are not terminated with a semicolon. Only the statements within these keyword statements are terminated with a semicolon.

Example

```
if RcvResult = 1 then
  LogMessage("receive no data condition occurred");
else if RcvResult = 2 then
  LogMessage("receive error condition occurred");
```

Expressions

An expression is a logical unit that the system evaluates.

Examples

```
RcvResult = 1;
MbxGetNextMsg(MsgId) != 0;
A + B
```

Operators

Operators define the simplest operation in an expression. This table lists the operators used in script commands:

Operator	Description
+	addition, concatenation
-	subtraction
*	multiplication
/	division
=	assignment, equality
>	greater-than
<	less-than
>=	greater-than or equal to
<=	less-than or equal to
!=	not equal to
!	logical not
&	logical and
|	logical or
<<	date modification

Chapter 2. Script Language Commands

ANSI Commands

AnsiClearRcvMsg

The AnsiClearRcvMsg command is used to receive data using the AnsiClear protocol. Data received is stored in a mailbox message with one attachment.

Syntax

```
AnsiClearRcvMsg();
```

AnsiClearSetRcvRsp

The AnsiClearSetRcvRsp command sets the response to send when receiving data. Not all systems require a response and the default is to not send any response.

Syntax

```
AnsiClearSetRcvRsp(string Response);
```

where:

- `Response` = The response to send. This can be one or more characters.

Example

```
AnsiClearSetRcvRsp("^0D");
```

AnsiClearSetSndRsp

The AnsiClearSetSndRsp command sets the response to expect when sending data. Not all systems will send a response and the default is not to expect any response.

Syntax

```
AnsiClearSetSndRsp(string Response);
```

where:

- `Response` = The response to expect. This can be one or more characters

Example

```
AnsiClearSetSndRsp("^0D");
```

AnsiClearSetTerm

The AnsiClearSetTerm command sets the termination character to be used for sending and receiving. The default is an ASCII carriage return (0x0D).

Syntax

```
AnsiClearSetTerm(string Terminator);
```

where:

- `Terminator` = The termination character

Example

```
AnsiClearSetTerm("^0D");
```

AnsiClearSndAtm

The AnsiClearSndAtm command is used to send a mailbox attachment using the AnsiClear protocol.

Syntax

```
AnsiClearSndAtm(integer MsgId, integer AtmId);
```

where:

- `MsgId` = Must contain the mailbox message identifier returned from `MbxGetNextMsg`.
- `AtmId` = Must contain the mailbox attachment identifier returned from `MbxGetNextAtm`.

Example

```
integer MsgId;
integer AtmId;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      AnsiClearSndAtm(MsgId, AtmId);
   end
end
```

ASCII Commands

AsciiRcvCtl

The AsciiRcvCtl command receives data using the ASCII protocol until the specified string or bytecount is found.

Syntax

```
AsciiRcvCtl(string StopRcv, [integer Timeout],[integer ContinueOnTimeout])
AsciiRcvCtl(integer ByteCount, [integer Timeout], [integer ContinueOnTimeout])
```

where:

- `StopRcv` = The control string used to indicate when to stop receiving
- `ByteCount` = The number of bytes to receive
- `Timeout` = Used to override the default. Optional. Defaults to 60 seconds.
- `ContinueOnTimeout` = Set to 1 if you wish the script to continue processing even though a timeout occurred before the stop receiving condition. Optional. Defaults to 0.

Example

```
//Receive data until the string "hello" is received
AsciiRcvCtl("hello");
AsciiRcvCtl("hello", 100);
AsciiRcvCtl("hello", 100, 1);
//Receive data 10 bytes of data
AsciiRcvCtl(10);
AsciiRcvCtl(10, 100);
AsciiRcvCtl(10, 100, 1);
```

AsciiRcvFile

The AsciiRcvFile command is used to receive data using the ASCII protocol. This command can be used in conjunction with SetRcvNoData and SetRcvError to look for other data responses in place of or lack of a receive file.

Syntax

```
integer AsciiRcvFile(string FileSpec, string EndOfFile, [integer Terminate]);
```

where:

- `FileSpec` = Fully qualified filespec pointing to a file. UNC naming can be used.
- `EndOfFile` = The characters that indicate the end of the file.
- `Terminate` = If set to 1, will terminate the receive if either the RcvNoData or RcvError conditions occur. Optional. Defaults to 0.

Return value

- 0 = If end of file condition occurred.
- 1 = If the RcvNoData condition occurred.
- 2 = If the RcvError condition occurred.

Example

```
integer RcvResult;
SetRcvNoData("no data");
SetRcvError("error");
RcvResult = AsciiRcvFile("c: emp\rcv.txt", "^04");
if RcvResult = 1 then
  LogMessage("receive no data condition occurred");
else if RcvResult = 2 then
  LogMessage("receive error condition occurred");
```

AsciiRcvMsg

The AsciiRcvMsg command is used to receive data using the ASCII protocol. Data received is stored in a mailbox message with one attachment. This command can be used in conjunction with SetRcvNoData and SetRcvError to look for other data responses in place of or lack of a receive file.

Syntax

```
integer AsciiRcvMsg(string EndOfFile, [integer Terminate]);
```

where:

- `EndOfFile` = The characters that indicate the end of the file.
- `Terminate` = If set to 1, will terminate the receive if either the RcvNoData or RcvError conditions occur. Optional. Defaults to 0.

Return value

- 0 = If end of file condition occurred.
- 1 = If the RcvNoData condition occurred.
- 2 = If the RcvError condition occurred.

Example

```
integer RcvResult;
SetRcvNoData("no data");
SetRcvError("error");
RcvResult = AsciiRcvMsg("^04");
```

```
if RcvResult = 1 then
  LogMessage("receive no data condition occurred");
else if RcvResult = 2 then
  LogMessage("receive error condition occurred");
```

AsciiSndAtm

The AsciiSndAtm command is used to send a mailbox attachment using the ASCII protocol.

Syntax

```
AsciiSndAtm(integer MsgId, integer AtmId,⌂string EndOfFile);
```

where:

- `MsgId` = Must contain the mailbox message identifier returned from MbxGetNextMsg.
- `AtmId` = Must contain the mailbox attachment identifier returned from MbxGetNextAtm.
- `EndOfFile` = Contains any data to be sent to denote end of file.

Example

```
integer MsgId;
integer AtmId;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      AsciiSndAtm(MsgId, AtmId, "^04");
   end
end
```

AsciiSndCtl

The AsciiSndCtl command transmits the specified string using ASCII protocol.

Syntax

```
AsciiSndCtl(string Data);
```

where:

- `Data` = The control string to be transmitted

Example

```
AsciiSndCtl("hello world");
```

AsciiSndFile

The AsciiSndFile command is used to send a file using the ASCII protocol.

Syntax

```
AsciiSndFile(string FileSpec, string EndOfFile);
```

where:

- `FileSpec` = A fully qualified filespec pointing to a file. UNC naming can be used.
- `EndOfFile` = The characters that indicate the end of the file.

Example

```
integer MsgId;
integer AtmId;
AsciiSndFile("c: emp\snd.txt", "^04");
```

Bisync Commands

BisyncAutoAnswer

The BisyncAutoAnswer command places the bisynchronous device into an auto answer mode waiting for incoming calls. This command should only be used in a host pool script for bisynchronous devices.

Syntax

```
BisyncAutoAnswer();
```

BisyncClose

The BisyncClose command should be issued after the completion of all BisyncRead or BisyncWrite commands to close the current transaction.

Syntax

```
BisyncClose();
```

Example

```
BisyncOpen(OPEN_READ_TEXT);
BisyncRcvMsg();
BisyncClose();
```

BisyncConfig

The BisyncConfig command allows you to load a configuration file that overrides the current configuration file.

Syntax

```
BisyncConfig(string ConfigFile);
```

where:

- `ConfigFile` = New configuration filename

BisyncDial

The BisyncDial command dials the phone number stored in the properties. This command allows you to load a new configuration file (by using the BisyncConfig) before dialing. If the system dialed automatically before calling BisyncConfig, the line would be dropped.

Syntax

```
BisyncDial();
```

Example

```
BisyncConfig("NewConfig");
BisyncDial();
```

BisyncEot

The BisyncEot command allows you to unconditionally clear the line by sending an end of transmission character and is usually preceded by a BisyncClose command.

Syntax

```
BisyncEot();
```

Example

```
BisyncOpen(WRITE_TEXT);
BisyncSndCtl("Logon");
BisyncClose();
BisyncEot();
```

BisyncOpen

The BisyncOpen command opens the line for sending or receiving data based on which constant is specified. When opening the line for receive, it waits for a line bid and responds with an acknowledgement. When opening the line for send, it bids the line and waits for an acknowledgement.

Syntax

```
BisyncOpen(integer Mode);
```

where:

- `Mode` = Open mode constant (see table below)

Constants

Constant	Description
`READ_TEXT`	Allows receiving non-transparent text data with translation. Transparent binary data is received without translation.
`READ_NO_TRANSLATE`	Allows receiving non-transparent text data without translation.
`READ_BINARY_TRANSLATE`	Allows receiving transparent binary data with translation.
`WRITE_TEXT`	Allows sending non-transparent text data with translation.
`WRITE_BINARY`	Allows sending transparent binary data without translation.
`WRITE_TEXT_NO_TRANSLATE`	Allows sending non-transparent text data without translation.
`WRITE_NO_IRS`	Allows sending non-transparent text data with translation but no record separators are included. This essentially packs the data into one record.
`WRITE_BINARY_TRANSLATE`	Allows sending transparent binary data with translation.
`WRITE_TEXT_BINARY`	Allows sending transparent binary data with translation but with special translation of special characters (CR/LF to RS).

Example

```
BisyncOpen(READ_TEXT);
BisyncRcvMsg();
BisyncClose();
```

BisyncRcvCtl

The BisyncRcvCtl command is used to receive data into an internal receive buffer that can be used to check its contents. It specifies other protocol characters that indicate when to stop receiving and then returns the value for the protocol character that ended the receive.

Syntax

```
Integer BisyncRcvCtl(integer EndRcv);
```

where:

- `EndRcv` = Protocol character indicating when to stop receiving:
  - 1 - Stops receiving data when an EOT isreceived.
  - 2 - Stops receiving data when an ETB is received.
  - 3 - Stops receiving data when an ETX is received.
  - 4 - Stops receiving data when an ETB or ETX is received. The return code depends on what ended the receive.

Return value

- 1 = An EOT was received.
- 2 = An ETB was received.
- 3 = An ETX was received.

Example

```
integer iResult;
iResult = BisyncRcvCtl(4);
if iResult = 1 then
   LogMessage("got EOT");
if iResult = 2 then
   LogMessage("got ETB");
if iResult = 3 then
   LogMessage("got ETX");
if RcvBufSearch("hello") = 1 then
   LogMessage("got hello");
```

BisyncRcvFile

The BisyncRcvFile command is used to receive a file using the Bisync protocol. This command can be used in conjunction with SetRcvNoData and SetRcvError to look for other data responses in place of or lack of a receive file.

Command format

```
integer BisyncRcvFile(string FileSpec);
```

where:

- `FileSpec` = A fully qualified filespec pointing to a file. UNC naming can be used.

Return value

- 0 = Normal end of file.
- 1 = If the RcvNoData condition occurred.
- 2 = If the RcvError condition occurred.

Example

```
integer RcvResult;
SetRcvNoData("no data");
SetRcvError("error");
RcvResult = BisyncRcvFile("c: emp\rcv.txt");
if RcvResult = 1 then
  LogMessage("receive no data condition occurred");
else if RcvResult = 2 then
  LogMessage("receive error condition occurred");
```

BisyncRcvMsg

The BisyncRcvMsg command is used to receive data using the Bisync protocol. Data received is stored in a mailbox message with one attachment. This command can be used in conjunction with SetRcvNoData and SetRcvError to look for other data responses in place of or lack of a receive file.

Syntax

```
integer BisyncRcvMsg();
```

Return value

- 0 = Normal end of file.
- 1 = If the RcvNoData condition occurred.
- 2 = If the RcvError condition occurred.

Example

```
integer RcvResult;
SetRcvNoData("no data");
SetRcvError("error");
RcvResult = BisyncRcvMsg();
if RcvResult = 1 then
  LogMessage("receive no data condition occurred");
else if RcvResult = 2 then
  LogMessage("receive error condition occurred");
```

BisyncSetEof

The BisyncSetEof command sets the end of termination character for both sending and receiving.

Command format

```
BisyncSetEof(string EofChar);
```

where:

- `EofChar` = End of file character

Example

```
BisyncSetEof("^26"); // set for ETB
```

BisyncSndAtm

The BisyncSndAtm command is used to send a mailbox attachment using the Bisync protocol.

Syntax

```
BisyncSndAtm(integer MsgId, integer AtmId);
```

where:

- `MsgId` = Must contain the mailbox message identifier returned from MbxGetNextMsg.
- `AtmId` = Must contain the mailbox attachment identifier returned from MbxGetNextAtm.

Example

```
integer MsgId;
integer AtmId;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      BisyncSndAtm(MsgId, AtmId);
   end
end
```

BisyncSndCtl

The BisyncSndCtl command transmits the specified string using Bisync protocol.

Syntax

```
BisyncSndCtl(string Data);
```

where:

- `Data` = Control string to be transmitted

Example

```
BisyncSndCtl("hello world");
```

BisyncSndCtlEtx

The BisyncSndCtlEtx command transmits the specified string using Bisync protocol and ends each block with an end of text (ETX).

Syntax

```
BisyncSndCtlEtx(string Data);
```

where:

- `Data` = Control string to be transmitted

Example

```
BisyncSndCtlEtx("hello world");
```

BisyncSndFile

The BisyncSndFile command is used to send a file using the Bisync protocol.

Syntax

```
BisyncSndFile(string FileSpec);
```

where:

- `FileSpec` = A fully qualified filespec pointing to a file. UNC naming can be used.

Example

```
BisyncSndFile("c: emp\snd.txt");
```

BisyncTable

The BisyncTable command allows you to load new translation tables. The new tables must be in the same format as the `asciiebc.ovr` and `ebcascii.ovr` files.

Syntax

```
BisyncTable(string Table);
```

where:

- `Table` = Filenames separated by a space character

Example

```
BisyncTable("newtbl.a2e newtbl.e2a");
```

DCL Commands

DclLogoff

The DclLogoff command performs a logoff sequence using the DCL protocol.

Syntax

```
DclLogoff();
```

DclLogon

The DclLogon command performs a logon sequence using the DCL protocol.

Syntax

```
DclLogon(string LogonAcct, string LogonId, string LogonPsw, string IEAcct,
string IEId, string IEPsw, string NewLogonPsw, string NewIEPsw);
```

where:

- `LogonAcct` = Advantis logon account.
- `LogonId` = Advantis logon identifier.
- `LogonPsw` = Advantis logon password.
- `IEAcct` = Information Exchange account.
- `IEId` = Information Exchange identifier.
- `IEPsw` = Information Exchange password.
- `NewLogonPsw` = New Advantis logon password. This is only used if you want to change your logon password.
- `NewIEPsw` = New Information Exchange password. This is only used if you want to change your Information Exchange password.

DclRcvMsg

The DclRcvMsg command is used to receive data using the DCL protocol. Data received is stored in a mailbox message with one attachment.

Syntax

```
DclRcvMsg([string UserMsgClass]);
```

where:

- `UserMsgClass` = Optional. If specified, will retrieve data that contains the specified user message classification.

DclSetAck

The DclSetAck command is used to set the type of acknowledgment messages you want to receive from Information Exchange.

Syntax

```
DclSetAck(string AckType);
```

where:

- `AckType` = Type of acknowledgment message you want to receive:
  - R = Receipt acknowledgment
  - D = Delivery acknowledgment
  - B = Both receipt and delivery

Example

```
DclSetAck("R");
```

DclSndAtm

The DclSndAtm command is used to send a mailbox attachment using the DCL protocol.

Syntax

```
AsciiSndAtm(integer MsgId, integer AtmId);
```

where:

- `MsgId` = Must contain the mailbox message identifier returned from MbxGetNextMsg.
- `AtmId` = Must contain the mailbox attachment identifier returned from MbxGetNextAtm.

Example

```
integer MsgId;
integer AtmId;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      DclSndAtm(MsgId, AtmId);
   end
end
```

Do Commands

DoRcv

The DoRcv command can be used to determine if a receive type session was requested (such as receive-only or send-receive). Because there is only one script per mailbox definition, this command provides an indication of the type of session that was requested so the appropriate commands can be issued.

Syntax

```
integer DoRcv
```

Return value

- 0 = If the session is not a receive only, or send and receive type session.
- 1 = If the session is a receive only, or send and receive type session

Example

```
if DoRcv then
begin
   // do something
end
```

DoSnd

The DoSnd command can be used to determine if a send type session was requested (such as send-only or send-receive). Because there is only one script per mailbox definition, this command provides an indication of the type of session that was requested so the appropriate commands can be issued.

Command format

```
integer DoSnd
```

Return value

- 0 = If the session is not a send only, or send and receive type session.
- 1 = If the session is a send only, or send and receive type session.

Example

```
if DoSnd then
begin
   // do something
end
```

EICON Commands

EiconCall

The EiconCall command is used to make an X.25 connection using Eicon Technology hardware.

Syntax

```
EiconCall();
```

EiconListen

The EiconListen command is used to listen for X.25 connections while using Advanced Data Distribution using Eicon Technology hardware.

Syntax

```
EiconListen();
```

EiconSetBICUG

The EiconSetBICUG command is used to set the X.25 facilities for bilateral closed user group selection. See the ITU (International Telecommunications Union) X.25 recommendations for a more in-depth description and value settings.

Syntax

```
EiconSetBICUG(integer Value1, integer Value2);
```

where:

- `Value1` = First and second digit index number to the bilateral closed user group.
- `Value2` = Third and fourth digit index number to the bilateral closed user group.

EiconSetClassNeg

The EiconSetClassNeg command is used to set the X.25 facilities throughput class negotiation. See the ITU (International Telecommunications Union) X.25 recommendations for a more in-depth description and value settings.

Syntax

```
EiconSetClassNeg(integer Value);
```

where:

- `Value` = Throughput class values.

Example

```
EiconSetClassNeg(119);
// sets throughput class negotiation to 1200
```

EiconSetCUG

The EiconSetCUG command is used to set the X.25 facilities closed user group selection. See the ITU (International Telecommunications Union) X.25 recommendations for a more in-depth description and value settings.

Syntax

```
EiconSetCUG(integer Value);
```

where:

- `Value` = Index number to the closed user group.

EiconSetCUGOA

The EiconSetCUGOA command is used to set the X.25 facilities closed user group with outbound access selection. See the ITU (International Telecommunications Union) X.25 recommendations for a more in-depth description and value settings.

Syntax

```
EiconSetCUGOA(integer Value);
```

where:

- `Value` = Index number to the outgoing closed user group.

EiconSetNUI

The EiconSetNUI command is used to set the X.25 facilities for network user identification. See the ITU (International Telecommunications Union) X.25 recommendations for a more in-depth description and value settings.

Syntax

```
EiconSetNUI(integer Length, string NUI);
```

where:

- `Length` = The length of the NUI string.
- `NUI` = The contents of the NUI with a maximum password length of 6 bytes and a maximum length of 8 bytes.

EiconSetPacketSize

The EiconSetPacketSize command is used to set the X.25 facilities packet size. See the ITU (International Telecommunications Union) X.25 recommendations for a more in-depth description and value settings.

Syntax

```
EiconSetPacketSize(integer Remote, integer Local);
```

where:

- `Remote` = Remote packet size code.
- `Local` = Local packet size code.

Example

```
EiconSetPacketSize(7,7);
// sets remote and local window size to 128 bytes
```

EiconSetRevFast

The EiconSetRevFast command is used to set the X.25 facilities for reverse charging, or fast select, or both. See the ITU (International Telecommunications Union) X.25 recommendations for a more in-depth description and value settings.

Syntax

```
EiconSetRevFast(integer Value);
```

where:

- `Value` = The value for reverse charging, or fast select, or both.

EiconSetUserData

The EiconSetUserData command is used to set the X.29 call user data parameters.

Syntax

```
EiconSetUserData(integer P1, integer P2, integer P3, integer P4, string UserData)
```

where:

- `P1` = Protocol ID 1
- `P2` = Protocol ID2
- `P3` = Protocol ID 3
- `P4` = Protocol ID 4

- `UserData` = User-defined data (maximum of 12 bytes)

Example

```
EiconSetUserData(192,0,0,0,"")
//sets all 4 protocol Ids and does not include user data
EiconSetUserData(192,0,0,0,"user data")
//sets all 4 protocol Ids and includes user data
```

EiconSetWindowSize

The EiconSetWindowSize command is used to set the X.25 facilities window size. See the ITU (International Telecommunications Union) X.25 recommendations for a more in-depth description and value settings.

Syntax

```
EiconSetPacketSize(integer Remote, integer Local);
```

where:

- `Remote` = Remote window size
- `Local` = Local window size

Example

```
EiconSetWindowSize(7,7);
// sets remote and local window size to 7
```

FTP Commands

FtpCD

The FtpCD command is used to change directories on the FTP server by using a CD command instead of the normal CWD command. This is used by GXS.

Syntax

```
FtpCD(string Directory);
```

where:

- `Directory` = The directory to change to on the FTP server.

FtpChangeDir

The FtpChangeDir command is used to change directories on the FTP server. The FtpChangeDir will issue a CWD command first, and if that fails and a 502 error is received, it will retry using the CD command.

Syntax

```
FtpChangeDir(string Directory);
```

where:

- `Directory` = The directory to change to on the FTP server.

FtpDelete

The FtpDelete command is used to delete a file on the FTP server.

Syntax

```
FtpDelete(string FileName);
```

where:

- `FileName` = The filename to delete on the FTP server.

FtpDoCmd

The FtpDoCmd command is used to issue RFC compliant FTP commands.

Syntax

```
integer FtpDoCmd(string Cmd, [string Arg]);
```

where:

- `Cmd` = The RFC compliant FTP command to issue.
- `Arg` = The FTP command arguments. Optional

Return value

Returns the value found at the beginning of the last line returned from the host, or a value greater than 500 if an error occurs. It is the user's responsibility to evaluate this code to determine if the command was successful.

Example

```
integer Result;
string[10] sResult;
string[20] sFinal;
Result = FtpDoCmd("RMD", "test");
ntoa(Result, sResult);
sFinal = "DoCmd=" + sResult;
logMessage(sFinal);
```

FtpGetDir

The FtpGetDir command is used to obtain a directory listing from the FTP server. This command is used in conjunction with FtpGetDirString to inspect each line of the directory output returned by the server.

Command format

```
integer FtpGetDir(integer bUseList);
```

where:

- `bUseList` = A boolean variable indicating the directory listing type. If the value is TRUE, a LIST command is sent to the server. If the value is FALSE, a NLST command is sent to the server.

Return value

The integer value returned indicates how many directory rows/lines were returned by the server.

Example

```
integer iCnt;
integer iIndex;
string[100] OneLine;
iCnt = FtpGetDir(LIST);  // TRUE=LIST  FALSE=NLST
iIndex = 0;
```

```
while iIndex < iCnt do
begin
   FtpGetDirString(iIndex, OneLine);
   // do something to find exact filename in the line
   FtpRcvMsg(OneLine);
   iIndex = iIndex + 1;
end
```

FtpGetDirString

The FtpGetDirString command is used to return each line of a directory output returned by the FTP server. A call to FtpGetDir must occur before this function is used, so that the directory output can be received.

Syntax

```
FtpGetDirString(integer iIndex, string buffer);
```

where:

- `iIndex` = The zero-based index of the directory output array.
- `buffer` = A string buffer to place the contents of that directory line, based on the array index.

Example

```
integer iCnt;
integer iIndex;
string[100] OneLine;
iCnt = FtpGetDir(LIST);  // TRUE=LIST  FALSE=NLST
iIndex = 0;
while iIndex < iCnt do
begin
   FtpGetDirString(iIndex, OneLine);
   // do something to find exact filename in the line
   FtpRcvMsg(OneLine);
   iIndex = iIndex + 1;
end
```

FtpHost

The FtpHost command is used to start a FTP server session.

Syntax

```
FtpHost();
```

Example

```
FtpHost();
SetStatus(SUCCESS);
```

FtpRcvFile

The FtpRcvFile command is used to receive a file using the FTP protocol. This command can be used in conjunction with SetRcvNoData and SetRcvError to look for other responses in place of, or lack of, a file.

Syntax

```
integer FtpRcvFile(string LocalFile, string RemoteFile, integer DeleteAfterRcv);
```

where:

- `LocalFile` = The name of the file to create on the local system.
- `RemoteFile` = The name of the file on the remote system.

- `DeleteAfterRcv` = Optional. Defaults to 0 or FALSE which does not delete the files. Set to 1 or TRUE to delete each file after the receive completes.

Return value

- 0 = If the receive completed successfully.
- 1 = If the RcvNoData condition occurred.
- 2 = If the RcvError condition occurred.

Example

```
FtpRcvFile("localfile.txt", "remotefile.txt", TRUE);
```

FtpRcvMsg

The FtpRcvMsg command is used to receive one file using the FTP protocol. Data received is stored in a mailbox message with one attachment.

Syntax

```
FtpRcvMsg(string filename, integer DeleteAfterRcv);
```

where:

- `FileName` = The filename to receive from the FTP server.
- `DeleteAfterRcv` = Optional. Defaults to 0 or FALSE which does not delete the files. Set to 1 or TRUE to delete each file after the receive completes.

Example

```
FtpRcvMsg("filename.txt", TRUE);
```

FtpRcvMsgAll

The FtpRcvMsgAll command is used to receive all files from the current directory on an FTP server. Data received is stored in a mailbox message with one attachment.

Syntax

```
FtpRcvMsgAll([integer DeleteAfterRcv]);
```

where:

- `DeleteAfterRcv` = Optional. Defaults to 0 or FALSE which does not delete the files. Set to 1 or TRUE to delete each file after the receive completes.

Example

```
FtpRcvMsgAll(TRUE);
```

FtpRename

The FtpRename command is used to rename a file on an FTP server.

Syntax

```
FtpRename(string OldName, string NewName);
```

where:

- `OldName` = Specifies the name of the existing file to be renamed.
- `NewName` = Specifies the new file name.

Example

```
FtpRename("oldfile", "newfile");
```

FtpSetMode

The FtpSetMode command is used to set the mode in which files are transferred to the FTP server.

Syntax

```
FtpSetMode(integer Mode);
```

where:

- `Mode` = Specify either ASCII mode or BINARY mode. The default is BINARY mode.

Example

```
FtpSetMode(ASCII);
```

FtpSndAtm

The FtpSndAtm command is used to send a mailbox attachment using the FTP protocol.

Syntax

```
FtpSndAtm(integer MsgId, integer AtmId,⌂string FileName);
```

where:

- `MsgId` = Must contain the mailbox message identifier returned from MbxGetNextMsg.
- `AtmId` = Must contain the mailbox attachment identifier returned from MbxGetNextAtm.
- `FileName` = Contains the name of the file to create on the FTP server.

Example

```
integer MsgId;
integer AtmId;
string[128] FileName;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      MbxGetAtmFileName(MsgId, AtmId, FileName);
      FtpSndAtm(MsgId, AtmId, FileName);
   end
end
```

FTPSndFile

The FTPSndFile command is used to send a file that is not in the mailbox system to an FTP server using the FTP transport.

Syntax

```
FtpSndFile (string LocalFilename, string RemoteFilename,integer DeleteFlag);
```

where:

- `LocalFilename` = This variable specifies the local filename of the file to send.
- `RemoteFilename` = This variable specifies the remote filename on the FTP server that should be created when the file is sent.
- `DeleteFlag` = Specifies whether the local file should be deleted after successfully sending the file. This variable must be specified as either TRUE or FALSE. There is no default.
  - TRUE = Delete the local file after successfully sending the file
  - FALSE = Do not delete the local file after successfully sending

GetSessionTry

The GetSessionTry command is used to determine which session attempt, initial or retry is currently running.

Syntax

```
integer GetSessionTry();
```

Return value

The return value is 0 if it is the initial session attempt. The return value is greater than 0 for each session retry.

Example

```
if GetSessionTry() = 0 then
   LogMessage("initial try");
else if GetSessionTry() = 1 then
   LogMessage("1st retry");
else if GetSessionTry() = 2 then
   LogMessage("2nd retry");
```

Kermit Commands

KermitRcvMsg

The KermitRcvMsg command is used to receive data using the Kermit protocol.

Syntax

```
KermitRcvMsg();
```

Example

```
if DoRcv then
begin
   AsciiSndCtl("D");
   AsciiRcvCtl("Your choice");
   AsciiSndCtl("Z");
   AsciiRcvCtl("File name?");
   AsciiSndCtl("*.*^0D");
   AsciiRcvCtl("Begin your transfer procedure...");
   KermitRcvMsg();
end
```

KermitSndAll

The KermitSndAll command is used to send all files using the Kermit protocol.

Syntax

```
KermitSndAll();
```

Example

```
if DoSnd then
begin
   AsciiSndCtl("U");
   AsciiRcvCtl("Your choice");
   AsciiSndCtl("Z");
   AsciiRcvCtl("File name?");
   AsciiSndCtl("*^0D");
   AsciiRcvCtl("Begin your transfer procedure...");
   KermitSndAll();
end
```

KermitSet8thBitQuote

The KermitSet8thBitQuote command specifies the character (in decimal notation) to be used as the 8th bit quoting character. Use this command to override the default 8thBitQuote value.

Command format

```
KermitSet8thBitQuote(integer 8thBitQuote);
```

where:

- `8thBitQuote` = Decimal representation of the character to be used as the 8th bit quoting character. Must be in the range (33 – 126). The default is 0 for not using 8th bit quoting.

KermitSetBlockCheckType

The KermitSetBlockCheckType command specifies the type of block check to use. Use this command to override the default BlockCheckType value.

Command format

```
KermitSetBlockCheckType(integer BlockCheckType);
```

where:

- `BlockCheckType` = Valid values are:
  - 1 = 1 byte checksum (default)
  - 2 = 2 byte checksum
  - 3 = 3 byte CRC

KermitSetBlockStart

The KermitSetBlockStart command specifies the character (in decimal notation) to be used as the block start. Use this command to override the default BlockStart value.

Syntax

```
KermitSetBlockStart(integer BlockStart);
```

where:

- `BlockStart` = Decimal representation of the character to be used as the block start. Must be in the range (0 – 127). The default is 1 (SOH).

KermitSetControlQuote

The KermitSetControlQuote command specifies the character (in decimal notation) to be used as the control quote. Use this command to override the default ControlQuote value.

Syntax

```
KermitSetControlQuote(integer ControlQuote);
```

where:

- `ControlQuote` = Decimal representation of the character to be used as the control quote. Must be in the range of (32 – 127). The default is 35 (#).

KermitSetEndOfLine

The KermitSetEndOfLine command specifies the character (in decimal notation) to be used as the end of line. Use this command to override the default EndOfLine value.

Syntax

```
KermitSetEndOfLine(integer EndOfLine);
```

where:

- EndOfLine - Decimal representation of the character to be used as the end of line. Must be in the range (0 – 127). The default is 13 (CR).

KermitSetNumberOfPads

The KermitSetNumberOfPads command specifies the number of pad characters to use. Use this command to override the default NumberOfPads value.

Syntax

```
KermitSetNumberOfPads(integer NumberOfPads);
```

where:

- `NumberOfPads` = Number of pad characters to use. Must be in the range (0 – 127). The default is 0 for none.

KermitSetPacketSize

The KermitSetPacketSize command specifies the maximum packet size to be used during file transfer. Use this command to override the default PacketSize value.

Syntax

```
KermitSetPacketSize(integer PacketSize);
```

where:

- `PacketSize` = Contains the maximum size of a Kermit data packet during file transfer. The default is 1024.

KermitSetPadCharacter

The KermitSetPadCharacter command specifies the character (in decimal notation) to be used as the pad character. Use this command to override the default PadCharacter value.

Syntax

```
KermitSetPadCharacter(integer PadCharacter);
```

where:

- `PadCharacter` = Decimal representation of the character to be used as the pad character. Must be in the range (0 – 127). The default is 0 (NULL).

KermitSetParity

The KermitSetParity command specifies that parity is in use and should be accounted for. Use this command to override the default Parity value.

Syntax

```
KermitSetParity(integer Parity);
```

where:

- `Parity` = 0 (zero) or 1. Specify 1 to inform the Kermit protocol that parity is in use and requires special processing. The default is 0 for no parity.

LogMessage

The LogMessage command is used to write a user specified string or message to the session log.

Syntax

```
LogMessage(string UserMsg);
```

where:

- `UserMsg` = A string value to be written to the session log.

Example

```
integer RcvResult
SetRcvNoData("no data");
SetRcvError("error");
RcvResult = AsciiRcvMsg("^04");
if RcvResult = 1 then
  LogMessage("receive no data condition occurred");
else if RcvResult = 2 then
  LogMessage("receive error condition occurred");
```

Mailbox (Mbx) Commands

MbxGetAtmContentType

The MbxGetAtmContentType command is used to retrieve the content type and content subtype from the attachment.

Syntax

```
MbxGetAtmContentType(integer AtmId, string ContentType, string ContentSubType);
```

where:

- `AtmId` = Must contain the attachment identifier for the attachment to be queried.
- `ContentType` = A string value used to return the content type.
- `ContentSubType` = A string value used to return the content subtype.

Example

```
integer AtmId;
integer Length1;
integer Length2;
string[100] SndCmd;
string[20]  ContentType;
string[20]  ContentSubType;
string[8]   UserClass;
SndCmd = "+SEND";
MbxGetAtmContentType(AtmId, ContentType, ContentSubType);  // format is
MsgClass_x[8] or DataFormat_x[8]
Length1 = len(ContentType);
Length2 = len(ContentSubType);
if strstr(ContentType, "MsgClass_") != -1 then
   UserClass = mid(ContentType, 9, Length1-9);
else if strstr(ContentSubType, "MsgClass_") != -1 then
   UserClass = mid(ContentSubType, 9, Length2-9);
SndCmd = SndCmd + " USERCLASS(" + UserClass + ")";
```

MbxGetAtmFileExt

The MbxGetAtmFileExt command is used to obtain the message attachment file extension. If the attachment contains `C:\TEST\TEXT.TXT`, this function returns TXT.

Syntax

```
MbxGetAtmFileExt(integer MsgId, integer AtmId, string FileExt);
```

where:

- `MsgId` = Must contain the mailbox message identifier returned from MbxGetNextMsg.
- `AtmId` = Must contain the mailbox attachment identifier returned from MbxGetNextAtm.
- `FileExt` = A string value that will be used to return the attachment file extension.

Example

```
integer MsgId;
integer AtmId;
string[20] FileExtension;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      MbxGetAtmFileExt(MsgId, AtmId, FileExt);
      // do something
   end
end
```

MbxGetAtmFileName

The MbxGetAtmFileName command is used to obtain the message attachment filename. If the attachment contains `C:\TEST\TEXT.TXT`, this function returns TEXT.TXT.

Syntax

```
MbxGetAtmFileName(integer MsgId, integer AtmId, string Filename);
```

where:

- MsgId = Must contain the mailbox message identifier returned from MbxGetNextMsg.
- AtmId = Must contain the mailbox attachment identifier returned from MbxGetNextAtm.
- Filename = A string value that will be used to return the attachment filename.

Example

```
integer MsgId;
integer AtmId;
string[128] FileName;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      MbxGetAtmFileName(MsgId, AtmId, FileName);
      // do something
   end
end
```

MbxGetAtmFilePath

The MbxGetAtmFilePath command is used to obtain the message attachment file path. If the attachment contains `C:\TEST\TEXT.TXT`, this function returns `C:\TEST\TEXT.TXT`.

Syntax

```
MbxGetAtmFilePath(integer MsgId, integer AtmId, string FilePath);
```

where:

- MsgId = Must contain the mailbox message identifier returned from MbxGetNextMsg.
- AtmId = Must contain the mailbox attachment identifier returned from MbxGetNextAtm.
- FilePath = A string value that will be used to return the attachment file path.

Example

```
integer MsgId;
integer AtmId;
string[128] FilePath;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      MbxGetAtmFilePath(MsgId, AtmId, FilePath);
      // do something
   end
end
```

MbxGetAtmFileTitle

The MbxGetAtmFileTitle command is used to obtain the message attachment file title. If the attachment contains `C:\TEST\TEXT.TXT`, this function returns TEXT.

Syntax

```
MbxGetAtmFileTitle(integer MsgId, integer AtmId, string FileTitle);
```

where:

- `MsgId` = Must contain the mailbox message identifier returned from MbxGetNextMsg.
- `AtmId` = Must contain the mailbox attachment identifier returned from MbxGetNextAtm.
- `FileTitle` = A string value that will be used to return the attachment file title.

Example

```
integer MsgId;
integer AtmId;
string[128] FileTitle;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      MbxGetAtmFileTitle(MsgId, AtmId, FileTitle);
      // do something
   end
end
```

MbxGetNextAtm

The MbxGetNextAtm command returns the next available mailbox attachment identifier that it ready to be sent.

Syntax

```
integer MbxGetNextAtm(integer AtmId)
```

where:

- `AtmId` = This variable is used as the Return value.

Return value

- Non zero = If a valid mailbox attachment identifier is available and ready to be sent.
- Zero = If no more mailbox attachments are available to send.

Example

```
integer MsgId;
integer AtmId;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      // do something
   end
end
```

MbxGetNextMsg

The MbxGetNextMsg command returns the next available mailbox message identifier that is ready to be sent.

Syntax

```
integer MbxGetNextMsg(integer MsgId)
```

where:

- `MsgId` = This variable is used as the Return value.

Return value

- Non zero = If a valid mailbox message identifier is available and ready to be sent.
- Zero = If no more mailbox messages are available to send.

Example

```
integer MsgId;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   // do something
end
```

MbxGetOriginalMsgId

The MbxGetOriginalMsgId command is used to retrieve the original message identifier from the message.

Syntax

```
MbxGetOriginalMsgId(integer MsgId, string OriginalMsgId);
```

where:

- `MsgId` = The mailbox message identifier for the message to be queried
- `OriginalMsgId` = A string value used to return the information.

Example

```
integer MsgId;
integer Length;
string[100] SndCmd;
string[10]  OriginalMsgId;
string[8]   MsgName;
string[5]   MsgSeqn;
SndCmd = "+SEND";
// code for reconciliation
MbxGetOriginalMsgId(MsgId, OriginalMsgId);
Length = len(OriginalMsgId);
if Length > 8 then
begin
   MsgName = left(OriginalMsgId, 8);
   MsgSeqn = mid(OriginalMsgId, 8, Length-8);
   SndCmd = SndCmd + " MSGNAME(" + MsgName + ") MSGSEQN(" + MsgSeqn + ")";
end
else
begin
   MsgName = OriginalMsgId;
   SndCmd = SndCmd + " MSGNAME(" + MsgName + ")";
end
```

MbxGetRcvrEmailAddr

The MbxGetRcvrEmailAddr command is used to retrieve the receiver gateway email address from the message.

Syntax

```
MbxGetRcvrEmailAddr(integer MsgId, string RcvrEmailAddr);
```

where:

- `MsgId` = The mailbox message identifier for the message to be queried.
- `RcvrEmailAddr` = A string value used to return the receiver's gateway email address.

Example

```
integer MsgId;
integer Slash;
integer Length;
integer Underscore;
string[100] SndCmd;
string[80]  RcvrEmailAddr;
string[8]   DestAcct;
string[8]   DestUserId;
string[3]   SysId;
SndCmd = "+SEND";
MbxGetRcvrEmailAddr(MsgId, RcvrEmailAddr);
// format is DESTACCT[8]_DESTUID[8]/SYSID[3] or LIST=X[8]
Length = len(RcvrEmailAddr);
if strstr(RcvrEmailAddr, "LIST=") >= 0 then
begin
   DestAcct = mid(RcvrEmailAddr, 5, Length-5);
   SndCmd = SndCmd + " LIST(" + DestAcct + ")";
end
else
begin
   Underscore = strstr(RcvrEmailAddr, "_");
   DestAcct = left(RcvrEmailAddr, Underscore);
   Underscore = Underscore + 1;
   Slash = strstr(RcvrEmailAddr, "/");
   // look for SYSID
   if ( Slash >= 0 ) then
   begin
      DestUserId = mid(RcvrEmailAddr, Underscore, Slash-Underscore);
      Slash = Slash + 1;
      SysId = mid(RcvrEmailAddr, Slash, Length-Slash);
      SndCmd = SndCmd + " SYSID(" + SysId + ")";
   end
   else
   begin
      DestUserId = mid(RcvrEmailAddr, Underscore, Length-Underscore);
      SndCmd = SndCmd + " DESTACCT(" + DestAcct + ") DESTUID(" + DestUserId + ")";
   end
end
```

MbxLogon

The MbxLogon command is used in Advanced Data Distribution to collect the mailbox ID and Advanced Data Distribution password.

Syntax

```
integer MbxLogon(string mailbox, string password);
```

where:

- `mailbox` = Advanced Data Distribution mailbox name
- `password` = Host password

Example

```
// Sample Pool Host Script
string[50] Mbx;
string[50] Psw;
// look for LOGON MBX=x PSW=x
AsciiRcvCtl("^0D");
if RcvBufSearch("LOGON") = 0 then
begin
   AsciiSndCtl("GOODBYE^0D");
   Exit();
end
// validate logon
ParseStart();
ParseNamed("MBX=", " ^0D", Mbx);
ParseNamed("PSW=", " ^0D", Psw);
if MbxLogon(Mbx, Psw) = 0 then
begin
   AsciiSndCtl("GOODBYE^0D");
   Exit();
end
```

MbxStartAtmLoop

The MbxStartAtmLoop command is used in conjunction with MbxStartMsgLoop, MbxGetNextMsg, and MbxGetNextAtm to iterate through all available mailbox attachments that are ready to send for a particular message. This command must precede MbxGetNextAtm and can only be used after establishing a message loop.

Syntax

```
MbxStartAtmLoop(integer MsgId);
```

where:

- `MsgId` = Must contain the mailbox message identifier returned from MbxGetNextMsg.

Example

```
integer MsgId;
integer AtmId;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      // do something
   end
end
```

MbxStartMsgLoop

The MbxStartMsgLoop command is used in conjunction with MbxGetNextMsg to iterate through all available mailbox messages that are ready to send. This command must precede MbxGetNextMsg.

Syntax

```
MbxStartMsgLoop();
```

Example

```
integer MsgId;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   // do something
end
```

OFT Commands

OftpAddSfidCheck

The OftpAddSfidCheck command adds the specified string to a table of valid originator IDs that will be checked against the sfidorig field of the SFID when receiving files. The file is rejected if the contents of the sfidorig field do not match any of the values in the table. This command must be issued for every ID that needs to be in the table and must proceed the OftpRemote or OftpHost script command.

Syntax

```
OftpAddSfidCheck(string CheckOriginator);
```

where:

- `CheckOriginator` = String value of the originator ID to be checked against the SFID sfidorig field.

Example

```
OftpAddSfidCheck("ID1");
OftpAddSfidCheck("ID2");
OftpRemote(OftpId, OftpPsw, OftpNewPsw);
```

See the Working with OFTP section in the *IBM® Sterling Gentran:Server® for Microsoft Windows Communications Gateway Configuration Guide* for more information about using the OFTP protocol.

OftpHost

The OftpHost command is used to perform Advanced Data Distribution functions when a trading partner initiates a communications session to an OFTP server. This command will take care of all sending and/or receiving, depending upon what type of session was requested, without specifying any of the mailbox-type commands.

Syntax

```
OftpHost(string SSIDcode, string SSIDpwsd, string SSIDuser,
[integer LogonUsingSSID]);
```

where:

- `SSIDcode` = A string value that will be sent in the ssidcode field of the SSID.
- `SSIDpwsd` = A string value that will be sent in the ssidpwsd field of the SSID.
- `SSIDuser` = A string value that will be sent in the ssiduser field of the SSID.
- `LogonUsingSSID` = Optional value. If set to TRUE, will use the incoming remote SSID information to log on to mailbox instead of the MbxLogon script command.

Examples

Example 1

This is an example of an OftpHost command.

```
AsciiSndCtl("IODETTE FTP READY ^0D");
OftpHost("SAMPLE ODETTE FTP HOST", "OFTP PSW", "");
SetStatus(SUCCESS);
```

Example 2

This is an example of an OftpHost command containing the optional LogonUsingSSID value.

```
AsciiSndCtl("IODETTE FTP READY ^0D");
OftpHost("SAMPLE ODETTE FTP HOST", "OFTP PSW", "", TRUE);
SetStatus(SUCCESS);
```

OftpNoEerps

The OftpNoEerps command instructs the system not to send any EERPs for files received and not to expect any EERPs for files sent.

Syntax

```
OftpNoEerps();
```

See the Working with OFTP section in the *IBM Sterling Gentran:Server for Microsoft Windows Communications Gateway Configuration Guide* for more information about using the OFTP protocol.

OftpRemote

The OftpRemote command is used to perform a complete remote session to an OFTP server. This command will take care of all sending and/or receiving, depending upon what type of session was requested, without specifying any of the mailbox-type commands. The optional parameters are used to verify that the ID and password information contained in the SSID is accurate. If a discrepancy is found, the session fails. The optional parameters must be used in conjunction with each other.

Syntax

```
OftpRemote(string OftpId, string OftpPsw,
string OftpNewPsw, [string IdToCheck, string PswToCheck]);
```

where:

- `OftpId` = Contains the OFTP user ID.

  Note: You can override the OftpID/SFID parameter on a per partner basis. To override the OftpID/SFID parameter, modify the default E-mail address in the partner profile by appending a / character, followed by the new SFID to use in the SFID header.
- `OftpPsw` = Contains the OFTP user password.
- `OftpNewPsw` = Contains the OFTP new user password. This is only used for certain systems that allow changing of the password. If used, it will be placed in the ssiduser field of the ssid structure.
- `IdToCheck` = Verifies that the OftpId contained in the SSID is accurate. Must be used in conjunction with PswToCheck.

- `PswToCheck` = Verifies that the OftpPsw contained in the SSID is accurate. Must be used in conjunction with IdToCheck.

See the Working with OFTP section in the *IBM Sterling Gentran:Server for Microsoft Windows Communications Gateway Configuration Guide* for more information about using the OFTP protocol.

OftpSetDynamicPassword

The OftpSetDynamicPassword command is used to dynamically update the OFTP password on both the local machine and the host machine, provided the host supports password changes. In order for this command to work properly, scriptvar-type variables must be used for the first three parameters of the OftpRemote command. There are three forms of this command which are shown in the examples below.

Syntax

```
OftpSetDynamicPassword([integer Switch], [integer PswExpire]);
```

- `Switch` = Optional. Integer value that identifies which field of the SSID to use for the new OFTP password.
  - If set to FALSE (default), the new password is sent in the ssiduser field of the SSID and the old password is sent in the ssidpwsd field.
  - If set to TRUE, the new password is sent in the ssidpwsd field of the SSID and the old password is sent in the ssiduser field.
- `PswExpire` = Optional. Integer value that identifies (in days) when the password expires. If this parameter is used, the Switch parameter must also be specified. When the password expires, a new password is generated automatically on behalf of the user.

See the Working with OFTP section in the *IBM Sterling Gentran:Server for Microsoft Windows Communications Gateway Configuration Guide* for more information about using the OFTP protocol.

Example

```
scriptvar string[25] OftpId;
scriptvar string[8] OftpPsw;
scriptvar string[8] OftpNewPsw;
AsciiRcvCtl("IODETTE FTP READY ^0D");

// Example1, OftpNewPsw is sent in the ssiduser field.
OftpSetDynamicPassword();
// same as OftpSetDynamicPassword(FALSE);

// Example2, OftpNewPsw is sent in the ssidpswd field.
OftpSetDynamicPassword(TRUE);

// Example3, after 30 days, a new password is generated
// automatically and sent in the ssiduser field.
OftpSetDynamicPassword(FALSE, 30);

// Once the session completes, OftpPsw is updated with
// the new password and OftpNewPsw is set to null.
OftpRemote(OftpId, OftpPsw, OftpNewPsw);
```

OftpSetEerpDelivered

An EERP that is received for a file sent is, by default, marked as PICKEDUP in the mailbox system. The OftpSetEerpDelivered()command overrides the default setting and marks the EERP as DELIVERED.

Syntax

```
OftpSetEerpDelivered();
```

See the Working with OFTP section in the *IBM Sterling Gentran:Server for Microsoft Windows Communications Gateway Configuration Guide* for more information about using the OFTP protocol.

Example

```
OftpSetEerpDelivered();
OftpRemote(OftpID, OftpPsw, OftpNewPsw);
```

OftpSetMaxRecordSize

The OftpSetMaxRecordSize command overrides the default record size of 0 (used with the default record format of U). This command is often used in conjunction with OftpSetRecordFormat.

Syntax

```
OftpSetMaxRecordSize(integer RecordSize);
```

where:

- `RecordSize` = Specifies the record size used with record formatting.

Example

```
OftpSetMaxRecordSize(80);  // sets 80 byte record size
OftpSetRecordFormat("F");    // sets fixed record formatting
```

OftpSetRcvDupCheck

When files are received during OftpRemote or OftpHost sessions, the OftpSetRcvDupCheck command checks for and rejects duplicate files based on the sfiddsn, sfiddate, sfidtime, and sfidorig field values of the SFID.

Syntax

```
OftpSetRcvDupCheck();
```

See the Working with OFTP section in the *IBM Sterling Gentran:Server for Microsoft Windows Communications Gateway Configuration Guide* for more information about using the OFTP protocol.

OftpSetRecordFormat

The OftpSetRecordFormat command overrides the default record format “U,” which is used for the entire communications session. If a record format override is specified in the content type of the message attachment, the system overrides the OftpSetRecordFormat command. OftpSetRecordFormat is often used in conjunction with OftpSetMaxRecordSize.

Syntax

```
OftpSetRecordFormat(string RecordFormat);
```

where:

- `RecordFormat` = Valid values are:
  - F
  - V
  - U

– T

Example

```
OftpSetRecordFormat("F");    // sets fixed record formatting
OftpSetMaxRecordSize(80);  // sets 80 byte record size
```

OftpSetSpecialEERP

The OftpSetSpecialEERP command specifies use of special end-to-end response packet processing where the destination and originator fields are swapped. This is normally used for TRADANET type networks.

Syntax

```
OftpSetSpecialEERP();
```

See the Working with OFTP section in the *IBM Sterling Gentran:Server for Microsoft Windows Communications Gateway Configuration Guide* for more information about using the OFTP protocol.

OftpSpecialLogicOff

The OftpSpecialLogicOff command is used to turn off the initial special logic setting used during negotiation. It applies only when using an X.25 connection and must be used before the X.25 connection is made. The remote site may still negotiate to use special logic.

Syntax

```
OftpSpecialLogicOff();
```

See the Working with OFTP section in the *IBM Sterling Gentran:Server for Microsoft Windows Communications Gateway Configuration Guide* for more information about using the OFTP protocol.

Parse Commands

ParseNamed

The ParseNamed command extracts a “chunk” from the buffer by looking for an identifying “name” in the data. It searches the whole buffer, regardless of the current parsing position. This allows the script to handle named parameters that can occur in any order. After extracting a chunk, the current parsing position will be advanced if the chunk was beyond the previous parsing position. This means that if you call ParseNext after calling ParseNamed several times, ParseNext will begin its search from the end of the named “chunk” that was furthest into the string.

Syntax

```
ParseNamed(string sName, string sEndChars, string sChunk);
```

where:

- `sName` = A string value to match in the data.
- `sEndChars` = A string of characters used to identify the end point of the search.
- `sChunk` = A string value to retrieve from the data.

Example

```
// Sample Script
string[50] Mbx;
string[50] Psw;
// look for LOGON MBX=x PSW=x
AsciiRcvCtl("^0D");
if RcvBufSearch("LOGON") = 0 then
begin
   AsciiSndCtl("GOODBYE^0D");
   Exit();
end
// validate logon
ParseStart();
ParseNamed("MBX=", " ^0D", Mbx);
ParseNamed("PSW=", " ^0D", Psw);
if MbxLogon(Mbx, Psw) = 0 then
begin
   AsciiSndCtl("GOODBYE^0D");
   Exit();
end
```

ParseNext

The ParseNext command extracts the next “chunk” of data from the buffer. It begins at the current parsing position and finds the start of the chunk by skipping over characters until it encounters a character that is not identified in the sSkipChars parameter. This command then locates the end of the chunk once it finds a character listed in the sEndChars parameter, and sets the current parsing position accordingly.

Syntax

```
ParseNext(string sSkipChars, string sEndChars, string sChunk);
```

where:

- `sSkipChars` = A string of characters to ignore in the buffer.
- `sEndChars` = A string of characters used to identify the end point of the search.
- `sChunk` = A string value to retrieve from the data

Example

```
// Sample Script
string[50] Command;
string[50] Mbx;
string[50] Psw;
// Reset the parser
ParseStart();
// Get the command, it is always at the start of
// the buffer
ParseNext("", "", Command);
ParseNamed("MBX=", " ^0D", Mbx);
ParseNamed("PSW=", " ^0D", Psw);
if MbxLogon(Mbx, Psw) = 0 then
begin
   AsciiSndCtl("GOODBYE^0D");
   Exit();
end
```

ParseSkip

The ParseSkip command advances the current parse position by nSkip characters.

Syntax

```
ParseSkip(number nSkip);
```

where:

- `nSkip` = The number of characters to skip.

ParseStart

The ParseStart command must be called prior to parsing each buffer from the remote computer. It resets the current parsing position to the start of the buffer.

Syntax

```
ParseStart();
```

Example

```
// Reset the parser and validate logon
ParseStart();
ParseNamed("MBX=", " ^0D", Mbx);
ParseNamed("PSW=", " ^0D", Psw);
if MbxLogon(Mbx, Psw) = 0 then
begin
   AsciiSndCtl("GOODBYE^0D");
   Exit();
end
```

Pause

The Pause command is used to pause execution of the script for the specified number of seconds.

Syntax

```
Pause(integer Seconds);
```

where:

- `Seconds` = Contains the number of seconds to pause.

RcvBufSearch

The RcvBufSearch command is used to search the last receive buffer for the specified string. This command is useful for receiving some control information and then determining what was received.

Syntax

```
integer RcvBufSearch(string SearchStr)
```

where:

- `SearchStr` = Contains the string to search for in last receive buffer.

Return value

- 0 = If the string buffer does not contain the search string.
- 1 = If the string buffer does contain the search string.

Example

```
AsciiRcvCtl("^0D");
if RcvBufSearch("hello" ) = 1 then
   LogMessage("found hello");
```

Set Commands

SetBlockSize

The SetBlockSize command is used to set or reset the blocksize used for various protocols.

Syntax

```
SetBlockSize(integer BlockSize);
```

where:

- `BlockSize` = Contains the value to set or reset the blocksize used during transfers.

SetBufferSize

The SetBufferSize command is used to reset the size of the internal read/write buffers. The default is 2048.

Syntax

```
SetBufferSize(integer BufferSize);
```

- `BufferSize` = Contains the new size of the internal read/write buffers.

SetRcvError

The SetRcvError command is used to specify a string to scan for during downloads using certain protocols to indicate an error condition. This is useful in situations where you might receive data or just a string indicating an error occurred.

Syntax

```
SetRcvError(string RcvErrorIndicator);
```

where:

- `RcvErrorIndicator` = A string value that indicates a receive error condition.

SetRcvFileMode

The SetRcvFileMode command controls whether the system overwrites, or appends to, files created locally.

Syntax

```
SetRcvFileMode(integer Mode);
```

where:

- `Mode` = Valid values are APPEND or OVERWRITE.

Example

```
SetRcvFileMode(APPEND);
SetRcvFileMode(OVERWRITE);
```

SetRcvNoData

The SetRcvNoData command is used to specify a string to scan for during downloads using certain protocols to indicate no data is available to receive. This is useful in situations where you might receive data or just a string indicating no data is available to receive.

Syntax

```
SetRcvNoData(string RcvNoDataIndicator);
```

where:

- `RcvNoDataIndicator` = A string value that indicates a receive no data condition.

SetSessionType

The SetSessionType command overrides the type of communications session the system performs. Use this command if the system cannot start communications with a particular session type.

Syntax

```
SetSessionType(integer Type);
```

where:

- `Type` - Value values are:
- 0 = Send Only
- 1 = Send/Rcv
- 2 = Rcv Only

Example

```
SetSessionType(0);
```

SetSpecialUpdate

The SetSpecialUpdate command overrides the communications session's normal operations. When this command is enabled, any calls to SndOk in the script are overridden and will not be marked sent until the `SetStatus(SUCCESS)` command is executed. Any data received is not sent to the mailbox system until the `SetStatus(SUCCESS)` command is executed. This command is useful when communicating to certain networks that use "session level," otherwise known as "all or nothing" logic.

Syntax

```
SetSpecialUpdate(integer TrueOrFalse);
```

where:

- `TrueOrFalse` = Boolean flag used to enable or disable this command.

SetStatus

The SetStatus command sets the session status. Use one of the predefined constant values either SUCCESS or FAILED.

Syntax

```
SetStatus( integer Status );
```

where:

- `Status` = Contains the value to set the session status (either SUCCESS or FAILED).

Example

```
SetStatus(SUCCESS);
```

SetTimeout

The SetTimeout command sets a new timeout value used during communications sessions.

Syntax

```
SetTimeout(integer Seconds);
```

where:

- `Seconds` = New timeout value in seconds (default 60)

Example

```
SetTimeout(60);
```

SndOK

The SndOK command is used to indicate a successful send of a mailbox message. This command must be issued once a message and all its attachments have been sent successfully so that the status indicators can be updated and so that the message will not be sent again. This command is also issued when an attachment has been sent successfully so that the status indicators can be updated and so that the attachment will not be sent again.

Syntax

```
SndOK(integer MsgId)
SndOK(integer MsgId, integer AtmId)
```

where:

- `MsgId` = The mailbox message identifier to update the status for. The mailbox message identifier for the attachment specified in AtmId.
- `AtmId` = The mailbox attachment identifier to update the status for.

Example

```
integer MsgId;
integer AtmId;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      AsciiSndAtm(MsgId, AtmId, "^04");
      SndOK(MsgId, AtmId);
   end
   SndOK(MsgId);
end
```

Tip Commands

TipRemote

The TipRemote command is used to perform a complete remote session to a Tradanet Network using the Tradanet Interface Protocol (TIP). To use this command, the user must first enable Tradanet commands through the mailbox configuration properties, select TIP commands, and then configure the options accordingly.

Syntax

```
TipRemote();
```

Example

```
TipRemote();
SetStatus(SUCCESS);
```

TipSetFwd

The TipSetFwd command sets the block forwarding characteristics used during a TipRemote session.

Syntax

```
TipSetFwd(integer Forwarding);
```

where:

- `Forwarding` = Valid values:
  - 0 - no forwarding character present
  - 1 - forwarding character present on input to service
  - 2 - forwarding character present on output from service
  - 3 - (default) forwarding character present on both input and output

Example

```
TipSetFwd(0);
TipRemote();
SetStatus(SUCCESS);
```

TipSetPadding

The TipSetPadding command sets the block padding characteristics used during a TipRemote session.

Syntax

```
TipSetPadding(string Padding);
```

where:

- `Padding` = Valid values are:
  - N - (default) No padding. All blocks use physical size
  - Y - Pad data blocks. Pad last block of data to blocksize.
  - A - Pad all blocks (directives and data) to blocksize.

Example

```
TipSetPadding("A");
TipRemote();
SetStatus(SUCCESS);
```

TipSetPadChar

The TipSetPadChar command sets the pad character used for block padding during a TipRemote session.

Syntax

```
TipSetPadChar(integer PadChar);
```

where:

- `PadChar` = Default 0. Character used to pad blocks to blocksize.

Example

```
TipSetPadChar(32);    // set to blank/space
TipRemote();
SetStatus(SUCCESS);
```

TipSetConType

The TipSetConType command sets the connection type used during a TipRemote session.

Syntax

```
TipSetConType(integer ConType);
```

where:

- `ConType` - Valid values are:
  - 1 - LAN
  - 2 - Default. X25
  - 3 - Other

Example

```
TipSetConType(3);
TipRemote();
SetStatus(SUCCESS);
```

WWA Commands

WwaLogoff

The WwaLogoff command performs a logoff sequence using the World Wide Async (WWA) protocol.

Syntax

```
WwaLogoff();
```

WwaLogon

The WwaLogon command performs a logon sequence using the World Wide Async protocol.

Syntax

```
WwaLogon(string LogonAcct, string LogonId, string LogonPsw, string IEAcct,
string IEId, string IEPsw, string NewLogonPsw, string NewIEPsw);
```

where:

- `LogonAcct` = :ogon account.
- `LogonId` = :ogon identifier.
- `LogonPsw` = :ogon password.
- `IEAcct` = Information Exchange account.
- `IEId` = Information Exchange identifier.
- `IEPsw` = Information Exchange password.
- `NewLogonPsw` = New logon password. This is only used if you want to change your logon password.

- `NewIEPsw` = New Information Exchange password. This is only used if you want to change your Information Exchange password.

WwaRcvMsg

The WwaRcvMsg command is used to receive data using the World Wide Async protocol. Data received is stored in a mailbox message with one attachment.

Syntax

```
WwaRcvMsg([string UserMsgClass]);
```

where:

- `UserMsgClass` = Optional. If specified, will retrieve data that contains the specified user message classification.

WwaSndAtm

The WwaSndAtm command is used to send a mailbox attachment using the World Wide Async protocol.

Syntax

```
WwaSndAtm(integer MsgId, integer AtmId);
```

where:

- `MsgId` = Must contain the mailbox message identifier returned from MbxGetNextMsg.
- `AtmId` = Must contain the mailbox attachment identifier returned from MbxGetNextAtm.

Example

```
integer MsgId;
integer AtmId;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      WwaSndAtm(MsgId, AtmId);
   end
end
```

Xmodem Commands

XmodemRcvFile

The XmodemRcvFile command is used to receive a file using the Xmodem protocol. The command SetRcvFileMode controls whether the system overwrites, or appends to, the file. The default is for the system to overwrite the file.

Syntax

```
XmodemRcvFile(string Filename);
```

where:

- `Filename` = The name of the file that is used for receiving data. Universal Naming Convention (UNC) names are valid.

Example

```
SetRcvFileMode(APPEND);
XmodemRcvFile("testfile.txt");
```

XmodemRcvMsg

The XmodemRcvMsg command is used to receive data using the Xmodem protocol. Data received is stored in a mailbox message with one attachment.

Syntax

```
XmodemRcvMsg();
```

XmodemSetFillChar

Thie XmodemSetFillChar command sets the fill character used in the Xmodem protocol to pad the last block of data if the data does not fill the entire block. The default fill character is 0 (NULL).

Syntax

```
XmodemSetFillChar(string FillChar);
```

where:

- `FillChar` = Specifies the character to use to fill the end of the last data block, if needed. Can be a hex value in the form of ^xx as shown in the example below.

Example

```
XmodemSetFillChar("^1A");
```

XmodemSndAll

The XmodemSndAll command is used to send all available message attachments as one file using the Xmodem protocol.

Because the command internally accomplishes some tasks, there is no need to:

- Define a loop for each message and attachment.
- Use the SndOk script command.

Syntax

```
XmodemSndAll();
```

Example

```
If DoSnd
   XmodemSndAll();
```

XmodemSndAtm

The XmodemSndAtm command is used to send a mailbox attachment using the Xmodem protocol.

Syntax

```
XmodemSndAtm(integer MsgId, integer AtmId);
```

where:

- `MsgId` = Must contain the mailbox message identifier returned from MbxGetNextMsg.

- `AtmId` = Must contain the mailbox attachment identifier returned from MbxGetNextAtm.

Example

```
integer MsgId;
integer AtmId;
MbxStartMsgLoop();
while MbxGetNextMsg(MsgId) != 0 do
begin
   MbxStartAtmLoop(MsgId);
   while MbxGetNextAtm(AtmId) != 0 do
   begin
      XmodemSndAtm(MsgId, AtmId);
   end
end
```

XmodemSndFile

The XmodemSndFile command is used to send a file using the Xmodem protocol.

Syntax

```
XmodemSndFile(string Filename);
```

where:

- `Filename` = The name of the file to be sent. UNC names can be used.

Example

```
XmodemSndFile("testfile.txt");
```

Zmodem Commands

ZmodemRcvFile

The ZmodemRcvFile command instructs the system on how to receive a file or multiple files that use the Zmodem protocol. When a file is received, by default it overwrites a pre-existing file. When used in conjunction with the SetRcvFileMode command, you can instruct the system to append the information received to a pre-existing file.

Syntax

```
ZmodemRcvFile(string FilenameOrPath [, integer PathIndicator]);
```

where:

- `FilenameOrPath` = Specifies either a fixed filename or a path. If using a fixed filename, all files received are written to this single file. If specifying a path name, a separate file is created in that location for each file received. Universal Naming Convention (UNC) names are valid entries.
- `PathIndicator` = Optional. The default setting is FALSE. If set to TRUE, you must specify a path name parameter.

Examples

Example 1

```
SetRcvFileMode(APPEND);
ZmodemRcvFile("testfile.txt");
// receives into a single file called testfile.txt
```

Example 2

```
ZmodemRcvFile"("c: emp\", TRUE);
// receives multiple files into the temp directory
```

ZmodemRcvMsg

The ZmodemRcvMsg command is used to receive data using the Zmodem protocol.

Syntax

```
ZmodemRcvMsg();
```

Example

```
if DoRcv then
begin
   AsciiSndCtl("D");
   AsciiRcvCtl("Your choice");
   AsciiSndCtl("Z");
   AsciiRcvCtl("File name?");
   AsciiSndCtl("*.*^0D");
   AsciiRcvCtl("Begin your transfer procedure");
   ZmodemRcvMsg();
end
```

ZmodemSndAll

The ZmodemSndAll command is used to send all files using the Zmodem protocol.

Syntax

```
ZmodemSndAll();
```

Example

```
if DoSnd then
begin
   AsciiSndCtl("U");
   AsciiRcvCtl("Your choice");
   AsciiSndCtl("Z");
   AsciiRcvCtl("File name?");
   AsciiSndCtl("*^0D");
   AsciiRcvCtl("Begin your transfer procedure");
   ZmodemSndAll();
end
```

ZmodemSndFile

The ZmodemSndFile command is used to send a file using the Zmodem protocol.

Syntax

```
ZmodemSndFile(string Filename);
```

where:

- `Filename` = The name of the file to be sent. UNC names can be used.

Example

```
ZmodemSndFile("testfile.txt");
```

Chapter 3. Script Language Functions

String Conditions and Functions

You can use string conditions in IF/THEN and IF/THEN/ELSE statements to perform comparisons between strings.

Examples of the syntax are as follows:

```
IF s1 = s2 THEN
IF s1 < s2 THEN
IF s1 > s2 THEN
```

The following string functions are also available for you to use:

- left
- right
- mid
- strdate
- concat
- strstr

left, right, mid syntax

The left, right, and mid functions enable you to extract substrings from a string. The left function extracts a specified number of characters from the left of the string variable or field and returns the result as a string. The right function extracts a specified number of character from the right of the string variable and returns the result as a string. The mid function extracts from a specified position in the string to the right, for a specified number of characters. This is an example of how the statements are used.

```
string[10] s;
string[3] s1;
string[3] s2;
string[4] s3;
string[7] s4;

s = "abcdefghij";
s1 = left(s,3);
s2 = right(s,3);
s3 = mid(s,3,4);
```

strdate syntax

The strdate function converts a datetime type into a string using a format that you specify. This function allows you to include static characters such as a slash (/), which gives you access to full date support.

```
datetime d;
string[8] s;

strdate(d,"%y/%m/%d",s);
```

concat syntax

The concat function concatenates a specified number of characters from one string onto the end of another string. The following example demonstrates the syntax to concatenate five characters from string "s2" onto the end of string "s1":

```
string[10] s1,s2;
concat(s1,s2,5);
```

strstr syntax

The strstr function finds a substring inside a string. This function returns the position of the first instance of the designated substring. If this function does not find the specified substring inside the string, it returns a value of -1.

```
integer d;

d = strstr("hello", "el");
```

Numerical Functions

The numerical functions enable you to convert one data type to another.

The following are the available numerical functions:

- len
- atoi
- aton
- ntoa

len syntax

The len function counts and returns the number of characters in a string.

```
integer a;
a = len("hello");
```

atoi, aton, ntoa syntax

The atoi function converts strings into integers.

The aton function converts string into real numbers.

The ntoa function converts integers and real numbers into strings.

```
integer a;
real b;
string[8] s;
a = atoi("5");
b = aton("5.5");
ntoa(5.5, s);
```

atoi

The atoi function is a numerical function that converts strings into integers. The numerical functions enable you to convert one data type to another.

Common use

The atoi function is often used with SQL maps where data in the database is stored as string types. It is also used after manipulating a string value that contains both alpha and numeric characters, to attain the numeric value.

Syntax

```
int = atoi(string);
```

where:

- `int` = integer variable
- `string` = string variable

Example

```
integer a;
string[20] s;
s = "5";
a = atoi(s);
// "a" contains the value 5
```

aton

The aton function is a numerical function that converts strings into real numbers. The numerical functions enable you to convert one data type to another.

Common use

The aton function is often used with SQL maps where data in the database is stored as string types. It is also used after manipulating a string value that contains both alpha and numeric characters, to attain the numeric value.

Syntax

```
real = aton(string);
```

where:

- `real` = real number variable
- `string` = string variable

Example

```
real a;
string[20] s;
s = "5.5";
a = aton(s);
// "a" contains the value 5.5
```

concat

The concat function concatenates a specified number of characters from one string onto the end of another string.

Common use

The concat function is used when values from two strings need to be concatenated together to form one string. It can also be used during debugging to create more detailed messages. For example:

```
String[50] msg;
msg =  "Field A: ";
concat(msg,#fielda,15);
```

```
messagebox(msg,0);
// Instead of outputting the contents of #fielda in a messagebox,
// it will output a label of "Field A:" along with the contents.
```

Syntax

```
concat(string,string,num_char);
```

where:

- `string` = string variable
- `num_char` = number of characters from the second string onto the end of the first string

Note: You may not use an ActiveX property as the first parameter of the concat function because the length of the property is unknown prior to compilation.

Example

```
string[20] s1,s2;
s1 = "IBM";
s2 = "Corporation";

concat(s1,s2,8);

//Concatenate eight characters from string "s2" onto the end of string "s1".
//s1 will now contain the value "IBM Corporat".
```

get

The get function enables you to access individual components of a datetime variable. Valid datetime components are year, month, day, hour, minute, second.

Common use

The strdate function was added to extended rules after the get function. The strdate function is used more often than the get function because it is more versatile.

Syntax

```
integer_variable = get datetime_component (datetime_variable);
```

where:

- `integer_variable` = integer variable
- `datetime_component` = individual component of the datetime variable
- `datetime_variable` = datetime variable of which you want to access a component part

Example

```
integer temp_days;
integer temp_hours;
datetime d;
temp_days = 0;
temp_hours = 0;
d = '12/25/2001 12:15:30';
//A fields value in the map can be assigned to the datetime variable
//"d" or a hard coded value can be assigned.

temp_days = get days (d);
 temp_hours = get hours (d);
```

```
//Accesses the days from the datetime variable "d" and loads into
//variable " temp_days ". Accesses the hours from the datetime
//variable "d" and loads into variable "temp_hours".
```

left

The left function extracts a specified number of character from the left side of a string variable or field and returns the result as a string.

Common use

The left function is used when you only need the first part of a string. If you only want the first five digits of a zipcode, use the following example:

```
#TEMP_ZIP = left(#ZIP_CODE,5);
```

The left function is also commonly used in conjunction with the len function, to "drop" characters from the end of a string. If you want to drop the last three characters of a string, use the following example:

```
#TEMP = left(#FIELD,len(#FIELD) – 3);
```

The right, left, and len functions can all be used together. If you want to remove 0s from the beginning of an ID, but there is not a set number of 0s, use the following example:

```
while left(#ID,1) = "0" do
#ID = right(#ID,len(#ID)-1);
```

Syntax

```
string_variable = left(string_variable,num_char);
```

where:

- `string_variable` = A variable defined as type string.
- `num_char` = integer variable

Note: You may not use an ActiveX property as the first parameter of the left function because the length of the property is unknown prior to compilation.

Example

```
string [25]name;
string [5]temp_variable;
name = "Acme Shipping Company"
temp_variable = left(name,4);
// "temp_variable" would contain "Acme"
```

len

The len function is a numerical function that counts and returns the number of characters in a string. The numerical functions enable you to convert one data type to another.

Common use

The len function is most often used inline with other functions, such as left and right. It is also used within a while/do loop to pad a string to a specific length. If you need to add 0s to the front of a string to make the string 10 characters, use the following example:

```
while len(#field) < 10 Do
   #field = "0" + #field;
```

Syntax

```
number_char = len(string);
```

where:

- `num_char` = integer variable
- `string` = The string you wish to evaluate.

Example

```
integer a;
a = 0;
a = len("hello");
// "a" contains the value 5
```

mid

The mid function extracts from a specified position in a string, either to the end of the string or for a specified number of characters and returns the resultant string. This function is zero-based.

Common use

The mid function is often used with the strstr function. The strstr function will return the position of a specific character, then the mid can be used to return a substring from that character.

Because the mid function is zero-based, it is often used incorrectly, with the resultant string off by one character. An easy way to determine the correct starting position is to envision a cursor and count how many times you need to move it to get to the starting position you want. See the example below.

Syntax

```
string_variable = mid(string_variable,start_pos,num_char)
```

where:

- `string_variable` = The variable containing the string you want to extract.
- `start_pos` = The starting position in the string of characters (integer).
- `num_char` = The number of characters from the starting position (integer).

Note: You may not use an ActiveX property as the first parameter of the mid function because the length of the property is unknown prior to compilation.

Example

```
string [25]name;
string [10]temp_variable;
name = "Acme Shipping Company"
temp_variable = mid(name,5,8);
//The map will read 8 characters in the string starting with
//the sixth character. It is essentially ignoring the first
//five characters, so "temp_variable" will contain "Shipping".
```

ntoa

The ntoa function is a numerical function that converts real numbers into strings. The numerical functions enable you to convert one data type to another.

Common use

The ntoa function is often used when you cannot change the data type for a field, such as when you are writing to a database. The ntoa function is also used to assist in debugging. For example, because you cannot use numeric fields in a messagebox, you must convert the value to a string first.

```
real b;
string[20] s, msg;
b = 5.5;
ntoa(b, s);
msg = "b: " + s;
messagebox(msg,0);
//The messagebox output will contain "b: 5.5"
```

Syntax

```
string = ntoa(real,string);
```

where:

- `real` = The real number variable you wish to convert.
- `string` = The name of the string in which you want to store the converted number.

Example

```
real b;
string[20] s;
b = 5.5;
ntoa(b,s);
//The variable "s" contains the string "5.5".
```

right

The right function extracts a specified number of characters from the right side of a string variable or field.

Common use

The right function is used when you only need the last part of a string. If you only want the last four digits of a social security number, use the following example:

```
#TEMP_SS = right(#SOCIAL,4);
```

The right function is also commonly used in conjunction with the len function, to "drop" characters from the beginning of a string. If you want to drop the first three characters of a string, use the following example:

```
#TEMP = right(#FIELD,len(#FIELD) – 3);
```

The right, left, and len functions can all be used together. For example, if you want to remove 0s from the end of an ID, but there is not a set number of 0s, use the following example:

```
while right(#ID,1) = "0" do
#ID = left(#ID,len(#ID)-1);
```

Syntax

```
string_variable = right(string_variable,num_char)
```

where:

- `string_variable` = The name of the string of characters you wish to manipulate.
- `num_char` = The number of characters to count from the right side of a string.

Note: You may not use an ActiveX property as the first parameter of the right function because the length of the property is unknown prior to compilation.

Example

```
string [25]name;
string [10]temp_variable;
name = "Acme Shipping Company"
temp_variable = right(name,7);
// "temp_variable" would contain "Company"
```

scriptvar

This command defines a script variable that can be used for editing in the user interface. Script variables are not required, but when used, they provide a user-friendly interface to the user for capturing specific information. By assigning a meaningful name to a script variable, users can readily determine what information needs to be provided. When you create a script variable, users are required to enter a value for that variable unless you have specified that the variable is optional by using the optional parameter.

Command format

```
scriptvar [optional] type name;
```

Parameters

where:

- `optional` = Identifies this script variable as an optional variable.
- `type` = Type of variable (integer, real, string, datetime or array).
- `name` = The name of the variable.

Return value

There are no return values for this command.

Example

```
scriptvar string[10] MailboxId;
scriptvar optional integer SomeNumber;
```

set

The set function enables you to define individual components of a datetime variable. Valid datetime components are year, month, day, hour, minute, second.

Common use

The date function was added to extended rules after the set function. The date function is used more often than set because it is more versatile.

Note: Setting an integer value higher than the logical limit for the component will increase the corresponding related component. For example, if you set a value of 14 for months, it will increase the year by 1 and use the value 2 for the months. If you use the value 80 for minutes, it will increase the hours by 1 and use 20 for the minutes.

Syntax

```
set datetime_component (datetime_variable,integer_variable);
```

where:

- `datetime_component` = The individual component of the datetime variable.
- `datetime_variable` = The datetime variable of which you want to access a component part.
- `integer_variable` = An integer variable

Example

```
integer a, b, c;
datetime d;
a = 5;
b = 3;
c = 11;
set months (d,a);
set days (d,b);
set hours (d,c);
//Defines the months of the datetime variable "d" from variable "a"
//Defines the days of the datetime variable "d" from variable "b".
//Defines the hours of the datetime variable "d" from variable "c".
```

strdate

The strdate function converts a datetime type into a string using a format that you specify. This function allows you to include static characters such as a slash (/), which gives you access to full date support.

Common use

The strdate function is often used when you cannot change the data type for a field, such as when you are writing to a database. The strdate function is also used to assist in debugging. Because you cannot use a date field in a messagebox, you must convert the value to a string first.

```
string[20] s, msg;

strdate(#datefield,"%m/%d/%Y",s);
msg = "Date: " + s;
messagebox(msg,0);
//The messagebox output will contain "Date: value"
```

The strdate function is also used to determine shipping days of the week, and adjust accordingly. For example, if you do not ship on Sundays, you can check if `%w` returns a 0 and if so, add a day to make it Monday.

```
String[10] shipday;

strdate(#ship_date,"%w",shipday);
if shipday = "0" then
  #ship_date = #ship_date << days(1);
```

Syntax

```
strdate(datetime,"format",string);
```

where:

- `datetime` = datetime variable (month specified as 1 - 12)
- `format` = desired date format (see Format specifiers, below)
- `string` = string variable

Example

```
datetime d;
string[8] s;
d = date(2012,4,6);
s="";

strdate(d,"%y/%m/%d",s);

//Converts a datetime variable into an eight character string in the
//format "year/month/day", in this case 2012/4/6.
```

Format specifiers

Table 1. Format specifiers

Format Specifier	Description
%8	ISO-8601 date format `YYYYMMDDTHHMMSS.mmmZ` Four-digit year, two-digit month, two-digit day, T (time) indicator, two-digit hour, two-digit minutes, two-digit seconds in Universal Time (also called Zulu Time or Greenwich Mean Time), Z (Zulu time) indicator (example: 20031209T123000.000Z) **Note:** This date format cannot be combined with any other format specifier.
%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%d	Day of the month as a decimal number (01 – 31)
%H	Hour in 24-hour format (00 – 23)
%I	Hour in 12-hour format (01– 12)
%j	Day of the year as a decimal number (001 – 366)
%m	Month as a decimal number (01 – 12)
%M	Minute as a decimal number (00 – 59)
%S	Second as a decimal number (00 – 59)
%U	Week of the year as a decimal number, with Sunday as the first day of the week (00 – 51)
%w	Weekday as a decimal number (0 – 6, with Sunday as "0")
%W	Week of the year as a decimal number, with Monday as the first day of the week (00 – 51)
%y	Year without the century as a decimal number (00 – 99)
%Y	Year with the century as a decimal number

Table 1. Format specifiers (continued)

Format Specifier	Description
%%	Percent sign

strstr

The strstr function finds a substring inside a string. This function returns the position of the first instance of the designated substring within the specified string. If this function does not find the specified substring in the string, it returns a value of -1. This function is zero-based.

Common use

The strstr function is often used with the mid function to return a substring. For example, if you wanted to extract a 10-digit PO number that is listed after a slash, use the following example:

```
String[10] po_number;
po_number = mid(#PONUM,strstr(#PONUM,"/"),10);
```

If you do not know the length of the substring, the strstr function can also be used to determine how many characters are to the right of the character, by subtracting the position returned by strstr from the length using len.

```
String[20] po_number;
po_number = mid(#PONUM,strstr(#PONUM,"/"),(len(#PONUM)–strstr(#PONUM,"/")));
```

Syntax

```
integer = strstr("string","substring");
```

where:

- `integer` = integer variable
- `string` = the string to evaluate
- `substring` = the part of the string you are interested in

Examples

```
integer d;
d = 0;
d = strstr("mississippi","is");
//Finds the first instance of the substring "is" inside the string
//"mississippi" and returns the position of that first substring.
```

To use this function to enable a purchase order number to be processed differently depending on its format (for example, if the third position of the purchase order number is numeric do "X," otherwise do "Y"), use the following example:

```
integer position;
string [1] PONumChar2;
PONumChar2=mid(#PONumber, 1, 1);
position=strstr("0123456789", PONumChar2);

//This function finds a substring within the string. So if the second
//position of purchase order number is not equal to -1 then it is
//numeric and "X" should be executed. Otherwise, the second position is
//not numeric and "Y" should be executed.
```

winexec

The winexec function enables you to execute another program while running the translator.

This program is executed asynchronously. You specify the program and determine how you want the program window displayed. You can also return an error code, if desired. If the error code is greater than 32, the program ran without errors. If the error code is less than 32, the program did not run because of an error. If the error code is "0," the system is out of memory. If the error code is "2," you didn't specify a file name. The error code is not the return value from the program you executed.

Notes:

- If you specify a program on another machine or in another domain, you must have the appropriate permission to access the specified folder.
- If translation is executing from an unattended process control command, the user ID under which that service is running must have the appropriate permission to access the specified file.

Syntax

```
winexec("program",window_display)
```

where:

- `program` = executable program name string (if necessary, including UNC or direct file path)
- `window_display` = number that indicates how you want the program window displayed (see Window display numbers, below)

Example

```
winexec("program.exe", 3)

//Exits Gentran:Server and executes the "program.exe" program asynchronously.
//The system displays the program window maximized (3).
```

Window display numbers

The window_display numbers that control the appearance of the program window are as follows (you must use the number to indicate how you want the program window displayed, not the window_display value):

Table 2. Window display numbers

Number	Window_Display	Definition
0	SW_HIDE	Hides the window and activates another window.
1	SW_SHOWNORMAL	Activates and displays a window. If the window is minimized or maximized, it is restored to the original size and position. This flag should be specified when displaying a window for the first time.
1	SW_NORMAL	Activates and displays a window in the original size and position.
2	SW_SHOWMINIMIZED	Activates the window and displays it as a minimized windows.

Table 2. Window display numbers (continued)

Number	Window_Display	Definition
3	SW_SHOWMAXIMIZED	Activates the window and displays it as a maximized window.
3	SW_MAXIMIZE	Maximizes the window.
4	SW_SHOWNOACTIVATE	Displays the window in its most recent size and position, but does not activate it (the current active window remains active).
5	SW_SHOW	Activates the window and displays it in its current size and position.
6	SW_MINIMIZE	Minimizes the window.
7	SW_SHOWMINNOACTIVE	Displays the window as a minimized window without activating it (the current active window remains active).
8	SW_SHOWNA	Displays the window in its most recent size and position without activating it (the current active window remains active).
9	SW_RESTORE	Activates and displays the window. If the window was minimized or maximized, it is restored to its original size and position.
10	SW_SHOWDEFAULT	Activates the window and allows Microsoft Windows to determine the size and position.

Chapter 4. Script Language Keywords

begin ... end

The begin/end keywords enclose a group of statements that form the body of an if/then/else statement or a while loop.

You can use the if/then keywords to execute one or more statements conditionally. If you include more than one statement in the body of an if/then loop, you must surround the statements with the begin/end keywords. If you only use a single statement, you can omit the begin and end.

Sterling Gentran:Server uses conditional logic to test conditions and then, depending on the results of the test, perform different operations. Conditions can be nested to any level.

Note: Do not end conditions with a semicolon (;) – this terminating syntax is necessary for statements only.

Syntax

```
if condition then
begin
     statement1;
     statement2;
end
```

Example

```
while MbxGetNextAtm(AtmId) != 0 do
   begin
      MbxGetAtmFileName(MsgId, AtmId, FileName);
      FtpSndAtm(MsgId, AtmId, FileName);
   end
```

break

The break keyword terminates the execution of the nearest enclosing while loop, and passes control to the statement that follows the end keyword. The break keyword is generally used in complex loops to terminate a loop before several statements have been executed.

Example

```
integer i
  i = 0;

while i<10 do
begin
    #Total = Total + 50;
    i = i + 1;
   if #Total > 100000
      Break;
   else
      continue;
end
//While the value contained in the variable "i" is less than ten,
//50 will be added to the field Total.
//If the value in the field Total becomes greater than 100000
//before "i" equals 10, break out of the while loop else continue
//processing until "i" equals 10.
```

continue

The continue keyword continues the execution of the innermost loop without processing the statements in the loop that follow the continue statement.

Example

```
integer i;

i = 0;
while i<10 do
begin
     i = i + 1;
     if (i = 8) then
          continue;
     if (i = 9) then
          break;
end
//As long as "i" has a value less than "10" the loop repeats.
//If "i" has a value of "8", the loop continues. If "i" has a
//value of "9" the loop terminates.
```

if ... then ... else

The if, then, and else keywords allow the use of conditional logic. Sterling Gentran:Server uses conditional logic to test conditions and then, depending on the results of the test, perform different operations.

Conditions can be nested to any level. You can use the if/then keywords to run one or more statements conditionally. The condition is typically a comparison, but it can be any expression that concludes with a numeric value. Sterling Gentran:Server interprets the value as either true or false. The system interprets a zero value as false and a nonzero value as true.

Sterling Gentran:Server evaluates the if/then condition, and if it is true, the system runs all the statements that follow the then keyword. If the condition is false, none of the statements following then are run.

You can use the else keyword in conjunction with if/then to define several blocks of statements, one of which is run. Sterling Gentran:Server tests the first if/then condition. If the condition is false, the system proceeds to test each sequential condition until it finds one that is true. The system runs the corresponding block of statements for the true condition. If none of the if/then conditions are true, the system runs the statements following the else keyword.

The begin/end keywords enclose a group of statements that form the body of an if/then/else statement. You can use the if/then/else keywords to run one or more statements conditionally. If you include more than one statement in the body of an if/then statement, you must surround the statements with the begin/end keywords. If you use only a single statement, you can omit the begin/end.

Note: Do not end conditions with a semicolon (;) – this terminating syntax is necessary for statements only.

Example

```
if condition then
begin
     statement1;
     statement2;
end
else condition then
begin
     statement3;
     statement4;
end
```

while ... do

The while ... do keywords run a statement repeatedly until the specified termination condition evaluates to zero. The system tests the terminating condition before each iteration of the loop, so a while loop executes zero or more times depending on the value of the termination expression.

The begin/end keywords enclose a group of statements that form the body of a while/do loop. You can use the begin/end keywords to run one or more statements conditionally. If you include more than one statement in the body of a while/do loop, you must surround the statements with the begin/end keywords. If you use only a single statement, you can omit the begin/end.

Note: Do not end conditions with a semicolon (;) – this terminating syntax is necessary for statements only.

Common use

The while/do function is often used to initialize arrays, add or remove characters from a string, and to loop through iterations of a repeating structure. For more in-depth examples, see the "Using While Do Loops In Extended Rules" white paper.

Syntax

```
while condition do
```

Example

```
integer i;

while i < 10 do
begin
       i = i + 1;
       if (i = 8) then
          continue;
       if (i = 9) then
          break;
end
//While "i" is less than ten, run the loop. If "i" is equal to or
//greater than ten, terminate the loop.
```

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing

Legal and Intellectual Property Law

IBM Japan Ltd.

19-21, Nihonbashi-Hakozakicho, Chuo-ku

Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be

Trademarks

Index

Product Number: 5725-D09

Printed in USA