BusinessLink Software Support

Event Services Guide

Version V2R1

This manual applies to Strategi version V2R1 and later and was last revised in February 2005.

ADVANCED BusinessLink Corp. may have patents and/or patent pending applications covering subject matter in this document. The furnishing of this document does not give any license to these patents.

Every effort has been made to ensure the accuracy of this manual. However, ADVANCED BusinessLink Corp. and Advanced BusinessLink (Australia) Pty. Ltd make no warranties with respect to this documentation, and shall not be liable for any errors, or for incidental or consequential damages in connection with the performance or use of this manual, or the examples pertaining to products and procedures as described herein. The information in this manual is subject to change without notice.

Other trademarks, trade names and brand and product names used in this manual are trademarks or registered trademarks of their respective holders.

Printed in the United States of America.

A Note to Readers

The latest versions of this document and other Technical Support Bulletins can be downloaded from ADVANCED BusinessLink Corp.'s Support Website, http://support.businesslink.com.

You may print this in duplex format using Adobe's Acrobat Reader, which is available for download from http://www.adobe.com/products/acrobat/readstep.html.

With some installs of Adobe Acrobat, your printer may not resolve the characters correctly, and once printed, all characters will appear as rectangles or as symbols. If this happens, you will need to select "Print as image" from the Acrobat print dialogue. This will cause the print to occur correctly.

If you have any questions, comments or suggestions, please feel free to contact BusinessLink Software Support via email at support@businesslink.com.

Sincerely,

BusinessLink Software Support

Table of Contents

Introduction

ADVANCED BusinessLink introduces managed Event Services to allow customer processing of data received onto the system by various Strategi services. Currently, Events for HTTP requests and file transfers via Pocket Strategi and Strategi/Remote are supported. However, the Event infrastructure supports expanding the number and type of system generated events for future implementation.

There are three entities related to Events: the Client, the Routing Entry and the Handler.

The Client always refers to a physical job running on the hosting system. In the case of distributed event mechanisms, the client may be acting as a proxy for another client on another system. Currently, no distributed event mechanism comparable to DHSM (WebCluster) has been implemented, but the infrastructure supports its addition.

The Routing Entry is used only by Strategi's internal event clients to determine which handler to invoke for the event. However, the routing entry is required for successful event processing.

The Handler is the customer written application that processes (handles) event data. It is possible for one program to be both handler and client, just as it is with HSM.

Relationship to HSM

The relationship of Events to HSM is one of complimentary technology. The Event infrastructure effectively removes all limits on the exchanged data size, but in order to support that, a more complex API with increased per transaction overhead was developed. The event handler can also be tied up for longer, processing data of an indeterminate size, than would an HSM server, such that a higher ratio of handlers to client may be necessary than with an HSM solution.

The Event infrastructure provides a stream connection to a customer program; HSM provides a transactional one. Each is optimized accordingly.

Limits and Restrictions

Due to restrictions based on the single-threaded nature of native programs, the sequence of data transmissions is strictly enforced. That is, a client cannot begin to receive the event response before it has completed sending all data and properties to the handler. Likewise, the handler cannot begin sending its response before it has received all properties and data from the client.

The Event infrastructure attempts to eliminate all restrictions on data size, or make the restriction so large as to be effectively unlimited.

The number of properties an event can have is not limited; however, properties are held in memory by the receiver, and the operating system may impose particular limits on the amount of memory the job can consume. The total size of each property keyword and value together is limited to 16Mb.

Because the keyword is used to look up a property, the property keyword is limited to a limited number of invariant characters: A-Z, 0-9, '+', '_' '.' and '-'. A comma should never be used in a keyword as this would make the property list obtained from *PROPERTYNAMES unusable. The API does not programmatically enforce restrictions on keywords because it would entail an undesirable performance penalty.

Property values are limited to “textual data” -- the property value is always translated for the client or handler upon receiving it. In order to send binary data using a property, it must be text encoded, for example, in hex.

The size and content of the event data is not limited in any way. But take note of any considerations due to translations discussed below.

Variable Length API

At the most basic level the Event API operates on zero-terminated strings. To allow for ease of use by the more traditional field oriented programming languages, such as RPG, COBOL and CL, an API layer was created to facilitate them. This layer accepts field data and lengths and translates them into zero-terminated strings for input parameters, doing the reverse for output parameters.

Translation Services

Single byte character sets and double byte character sets are referred to as SBCS and DBCS, respectively. CCSID is an IBM term referring to Coded Character Set Identifier. It denotes a character set and an encoding scheme. The Event API supports only pure CCSIDs -- that is, CCSIDs composed entirely of SBCS or DBCS and require no special encoding rules. These CCSIDs use exactly one (SBCS) or two (DBCS) bytes always to represent a given character.

The following table lists the special values for common CCSIDs.

Value	CCSID	Meaning
*CURRENT	*	CCSID of the job when the Set Option API is called.
*JOB	*	CCSID of the job when a given translation is done.
*UNICODE	13488	Unicode or UCS double byte character set.
*HEX	65535	Indicates the data is binary and cannot be translated.

Table 1: Common CCSID Special Values.

Both data and properties may be translated according to the requirements of the client and handler. By default the CCSID used is that of the job using the API. The CCSID can be set separately for properties and data.

Using *CURRENT is slightly more efficient than *JOB. In most circumstances, *CURRENT produces the same effect because few jobs change their CCSID partway through. Using *JOB causes the current CCSID to be retrieved from the job's attributes every time a translation is performed.

Only the data can be set to use a CCSID of *HEX -- property values are considered to be textual elements. If either the client or handler sets *HEX for the data CCSID, no translation is possible.

Translation, when done, is always done by the receiver of the data. This allows the greatest efficiency when both client and handler are using the same CCSID.

Component Diagrams

The following component diagrams show the interface components of an event, the data flow between client and handler, and native API layering.

The interface of components in an event can be represented as follows:

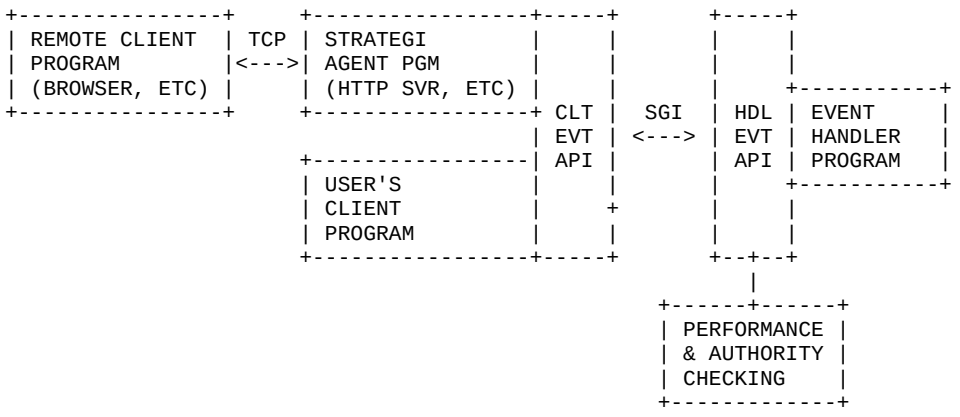

Figure 1: Event Interface Components.

The flow of data between the client and handler can be represented as follows:

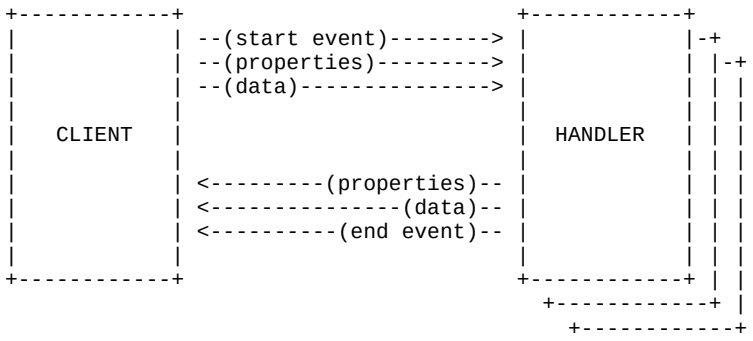

Figure 2: Data Flow Between Client and Handler.

Once the event is started, all subsequent communications until the event is ended take place with the same event handler instance. Subsequent events will use the next available handler.

The layering of the native APIs can be represented as follows:

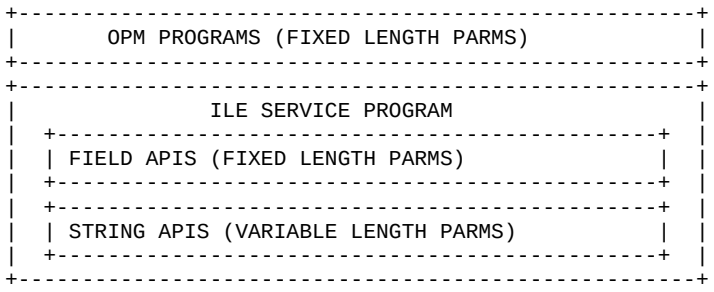

Figure 3: Native API Layering.

The API for Java is documented separately using javadoc, and the code is not layered on top of this native API.

Access Control

Access to an event handler can be controlled on a per-user basis at either the handler or opcode level. Permission to use a handler or opcode is defined by a combination of client type, client identification, handler, and opcode.

The following table shows valid access combinations that may be defined, in the order checked. If no access record is found for a given user, the default access value is *NO.

Client Type	Client ID	Handler	Opcode
X	X	X	X
X	X	X	-
X	X	-	-
X	-	X	X
X	-	X	-
X	-	-	-
-	-	X	X
-	-	X	-
-	-	-	-

Table 2: Access Control Combinations.

The handler has access to information that identifies the client to it. This information is available following the Receive Event API in the system properties *CLIENTTYPE, *CLIENTID, *CLIENTLOCATION, *USERATTRIBUTE, *AUTHORITYTYPE, and *AUTHORITYID.

Performance Monitoring

The time taken for each event handler to process each opcode is recorded, with fastest, slowest and average times. These are available using the xxxEVTPFR (Event Performance) commands, or option 22 from WRKEVTHDL. Note that this is the time for the handler to process the event, not the round-trip time for the client, which may include other communications, etc.

Error Handling

All Event APIs return a boolean indicator for success/failure. For the string APIs (intended for C, C++ and similar languages), the return value of the function is used. For field oriented APIs, a 1-character additional parameter is used. If an API fails, the details are available using the Get Property API for properties *ERRORCODE, *ERRORTEXT and *ERRORDETAIL. The Get Property API can only fail if the target space for the property value is too small.

Handler Interfaces

When designing the event handler, the sequence in which the APIs must be called is:

1. Receive Event
2. Receive Data or Store
3. Send Property
4. Send Data
5. End Event

Handler Pseudo Code

```
while not stopped {
    Receive Event
      if error occurred goto END
      if opcode is "*STOP" then set stopped flag to true
    Receive Data
      if error occurred goto END
    Send Properties
      if error occurred goto END
    Send Data

    END:
      log/report error if not blank
    End Event
    }
```

Handler APIs

The following table lists ILE field-oriented, string-oriented and OPM program Event Handler APIs.

Short Description	ILE Field API	ILE String API	OPM Program API
Receive Event	EVHRCVEVT	evh_rcvevt	EVTHDLRE
Get Property	EVHGETPRP	evh_getprp	EVTHDLGP
Receive Data	EVHRCVDTA	evh_rcvdta	EVTHDLRD
Store	EVHSTORE	evh_store	EVTHDLST
Send Property	EVHSNDPRP	evh_sndprp	EVTHDLSP
Send Data	EVHSNDDTA	evh_snddta	EVTHDLSD
End Event	EVHENDEVT	evh_endevt	EVTHDLEE
Set Option	EVHSETOPT	evh_setopt	EVTHDLSO

Table 3: Handler APIs.

Of these, only the Receive Event and End Event are required for processing an event. Set option may be called at any time, but when the option will take effect depends on the current state of the API. Get property may be called at any time, but will only return a non-blank value when called between Receive Event and End Event, except for the error properties which will be set if Set Option fails.

The string oriented APIs expect all text strings as zero-terminated (also known as null-terminated, which means the last character is hex 00 for SBCS and hex 0000 for DBCS). The field oriented APIs, those used in RPGLE and CLLE programs (for example) contain additional parameters to specify the lengths of certain parameters. In addition these APIs pass a field for the return value. Fields which are only used by the field APIs are marked with a double asterisk (**) following.

Important to note is that all text parameters are variable length. Text parameters with no effective maximum length are denoted with a length of asterisk (*). For text parameters with a maximum length, the length shown does not include the zero-terminator when the string APIs are used. That if the length is shown as 10, and you are using a string API you will actually have a field of 11 characters including the terminator. All maximum lengths are subject to change in future releases of Strategi.

All numeric parameters are four byte integers for the string APIs and decimal 15.5 for the field APIs. These are noted in the parameter lists by using the place-holder N.

All parameters lengths are followed by a single character 'I' or 'O', to indicate whether the parameter is an input or output parameter. No parameters are both input and output.

Receive Event

API (field, string, OPM):
EVHRCVEVT, evh_rcvevt, EVTHDLRE
Purpose:
Waits for an event to occur.

Parameters:

Opcode	10	O	Opcode for the event.
Opcode Length	N	I	Supplied length of the Opcode parameter.
Timeout	N	I	Number of seconds to wait for an event before returning.
Success Flag**	1	O	Return flag; 1=success, 0=failure.

Opcode allows one handler program to process many events without being directly dependent upon the event code, or upon other criteria used by Strategi in the future, to recognize an event. The meaning of an Opcode is logically defined by the handler.

Once an event has occurred the handler must process it. If the event is not processed, any subsequent attempt to call this API will result in an immediate error. Calling the End Event API will always reset the state of the API so that a Receive Event call is valid, except following the *STOP Event.

The *STOP Event is used to signal the handler to exit. Correct event processing should be done, following which the handler should exit its main control loop and go to end.

Get Property

API (field, string, OPM):
EVHGETPRP, evh_getprp, EVTHDLGP
Purpose:
Retrieves a property value. Property keywords beginning with an asterisk (*) denote a property supplied by Strategi. All other properties are set by the client.

Parameters:

Keyword	*	I	Name of the property.
Keyword Length**	N	I	Length of the property parameter.
Value	*	O	Variable to return the property value into.
Value Length	N	O	Maximum length of the value parameter. If the property value is longer than this, an error results.
Success Flag**	1	O	Return flag; 1=success, 0=failure.

If the current property value does not fit an error is generated. If the property is not defined (that is it was not transmitted by the client) then a blank value is returned. This means that sending a blank property and not sending the property are logically equivalent, allowing a client to avoid unnecessarily using communications bandwidth.

The Strategi HTTP event client sends all HTTP headers as properties beginning with the string "header.", and all URL data values as properties beginning with "urldata.". It also sends other useful information about the request that triggered the event.

All Strategi client programs send the *SOURCE and *EVENT properties.

Special Properties	Description
*CLIENTTYPE	The type of client that initiated the event. Possible values: *STRATEGI, *HOSTUSER, *HANDLER
*CLIENTID	The client identification string. This is dependant on the client type. For *STRATEGI it is the user number; for *HOSTUSER it is the user profile.
*CLIENTLOCATION	The peer system name that the client is located on. Currently this parameter is always *LOCAL A future version of Strategi will allow events to be actioned on a system which is different from the originating event.
*USERATTRIBUTE	The value of the user attribute named in the handler configuration for the user which triggered the event, if the client type is *STRATEGI.
*AUTHORITYTYPE	The type of client that was checked for authority to the handler. For single step events this is always the same as the client type. For multiple step events it is the type of the immediately previous client in the chain.
*AUTHORITYID	The ID of the client that was checked for authority to the handler. For single step events this is always the same as the client ID. For multiple step events it is the ID of the immediately previous client in the chain.
*SYSTEMPROPERTYNAMES	A comma delimited list of all available system properties. These properties are resolved internally to the event API and available and valid for all events. These properties all begin with an asterisk (*).
*PROPERTYNAMES	A comma delimited list of all properties received from the client program. These may include properties beginning with an asterisk when the client is a Strategi program, but typically will not.
*ERRORCODE	The abstract high-level error code when an error has occurred, or blank to indicate no error. This should always be tested immediately before calling the end event API. For the handler, the only value can be *ERROR, meaning an error occurred using the event API.
*ERRORTEXT	A text description of the error, suitable for display to an end user, and in the format "SGInnnn Xxxx...". The Strategi message ID can be reliably interrogated for specific behavior, the text string following cannot be -- it may change from release to release.
*ERRORDETAIL	An error specific detail message that should be logged or displayed for each error to assist in problem resolution.
*SOURCE	The source of the event (only from Strategi event clients). This can be *HTTP, *PSTRATEGI or *REMOTE.
*EVENT	The event code specified by the client, which Strategi uses in looking up the routing entry to arrive at the handler and opcode.
*REFERENCE	The reference number of the file transfer if the event originated from *PSTRATEGI or *REMOTE.
*GROUP	Transfer Group used during file transfer to/from the host in Pocket Strategi. This applies to *PSTRATEGI.
*TRACKINGNUMBER	Transfer Group tracking number used. This applies to *PSTRATEGI.
*PACKEDLENGTH	Packed file length. This applies to *PSTRATEGI.
*UNPACKEDLENGTH	Original file length. This applies to *PSTRATEGI.
*FILENAME	Original file name (last part of the path). This applies to

	*PSTRATEGI.
*FILEPATH	Original file path (with '/' or '\' converted to native , but with X: from DOS remaining if that was given on the original file name. This applies to *PSTRATEGI.

Table 4: Handler Special Properties.

Receive Data

API (field, string, OPM):
EVHRCVDTA, evh_rcvdta, EVTHDLRD
Purpose:
Receives a block of data from the Event API. This API is called repeatedly until it receives a data length of -1, which indicates end of data.

Parameters:

Data Block	*	O	Variable for returning the data into.
Data Maximum	N	I	Maximum length of the Data Block parameter.
Data Length	N	O	Pointer to a numeric entity to return the amount of data actually read.
Success Flag**	1	O	Return flag; 1=success, 0=failure.

The maximum number of bytes (not characters) put into data block is determined by the provided data maximum parameter. The number of bytes actually returned is returned in the data length parameter. For a receive CCSID which is a DBCS this represents half as many characters.

In CL and RPG all parameters are passed by reference, that is using pointers, so the term pointer may be unfamiliar to programmers of these languages. For these languages, a pointer parameter is simply a variable like any other in the program.

Store

API (field, string, OPM):
EVHSTORE, evh_store, EVTHDLST
Purpose:
Provides a convenient and easy way to store the relevant data to an IFS file.

Parameters:

File Name	*	I	Name of the IFS file to write the data to.
File Name Length**	N	I	Length of the File Name parameter.
Success Flag**	1	O	Return flag; 1=success, 0=failure.

It will store the data according to what the client sends. This may or may not contain information in addition to the file contents.

If the file already exists it will be silently over-written.

Send Property

API (field, string, OPM):
EVHSNDPRP, evh_sndprp, EVTHDLSP
Purpose:
Sends a property to the client as part of the event response. This API is called repeatedly, once for each property.

Parameters:

Keyword	*	I	Property keyword.
Keyword Length**	N	I	Length of the Keyword parameter.
Value	*	I	Property value.
Value Length**	N	I	Length of the Value parameter.
Success Flag**	1	O	Return flag; 1=success, 0=failure.

Properties beginning with an asterisk (*) are reserved for Strategi and will be rejected if used by customer code.

Send Data

API (field, string, OPM):
EVHSNDDTA, evh_snddta, EVTHDLSD
Purpose:
Sends a block of data to the client as part of the event response. This API is called repeatedly until all data is sent.

Parameters:

Data Block	*	I	Data to send.
Data Length	N	I	Length of the data in the Data Block parameter.
Success Flag**	1	O	Return flag; 1=success, 0=failure.

The API will send all of the data requested before returning. If it is unable to send the data, an error is indicated and the event is aborted - the handler should log the error, call End Event and proceed to wait for another event.

End Event

API (field, string, OPM):
EVHENDEVT, evh_endevt, EVTHDLEE
Purpose:
Ends event processing and resets all variables, clears properties and prepares to receive another event.

Parameters:
None

The End Event API can never fail.

This API is permitted at any time; if called while processing an event which is not in error any pending data will be received and appropriate responses made to the client.

If the opcode was *STOP, the handler should not proceed to receive another event, but instead terminate after calling this API.

Set Option

API (field, string, OPM):
EVHSETOPT, evh_setopt, EVTHDLSO
Purpose:
Allows optional API behavior to be specified and the setting of individual preferences and requirements.

Parameters:

Keyword	*	I	Keyword of the option to set.
Keyword Length**	N	I	Length of the Keyword parameter.
Value	*	I	Value to set; exact content depends on keyword.

Value Length**	N	I	Length of the Value parameter.
Success Flag**	1	O	Return flag; 1=success, 0=failure.

The following table lists valid options and associated values.

Valid Options	Description/Values
CCSID	CCSID to use for all data. Values: Any pure CCSID, *UNICODE, *JOB, *CURRENT. *JOB indicates to use the CCSID of the job as it is when each translation is done. *CURRENT indicates to get the job CCSID as it currently is and set it as the CCSID. The default is the CCSID of the job when the event API is initialized.
TEXTCCSID	Set only the text (properties, opcode, etc) CCSID. Values and default as for CCSID.
SNDDTACCSID	Set only the CCSID for data that is sent. Default as for CCSID. Values: Any pure CCSID, *UNICODE, *JOB, *CURRENT, *HEX. *HEX indicates that no translation is done - the binary data is delivered exactly as sent.
RCVDTACCSID	Set only the CCSID for data that is received. Default as for CCSID. Values: Any pure CCSID, *UNICODE, *JOB, *CURRENT, *HEX. *HEX indicates that no translation is done - the binary data is delivered exactly as sent.
TIMEOUT	Set the communications send/receive timeout in seconds. Valid values are 1-2Gb in seconds. Default value is 60 seconds.
SNDTIMEOUT	Set the communications send timeout in seconds. See TIMEOUT.
RCVTIMEOUT	Set the communications receive timeout in seconds. See TIMEOUT.

Table 5: Handler Valid Options and Values.

Client Interfaces

Client Pseudo Code

```
...
Start Event
  if error occurred goto END
Send Properties
  if error occurred goto END
Send Data
  if error occurred goto END

Receive Response
  if error occurred goto END
Receive Data
  if error occurred goto END

END:
  log/report error if not blank
End Event
...
```

Client APIs

The following table lists ILE field-oriented, string-oriented and OPM program Event Client APIs.

Short Description	ILE Field API	ILE String API	OPM Program API
Start Event	EVCSTREVT	evc_strevt	EVTCLTSE

Send Property	EVCSNDPRP	evc_sndprp	EVTCLTSP
Send Data	EVCSNDDTA	evc_snddta	EVTCLTSD
Receive Response	EVCRCVRSP	evc_rcvrsp	EVTCLTRR
Get Property	EVCGETPRP	evc_getprp	EVTCLTGP
Receive Data	EVCRCVDTA	evc_rcvdta	EVTCLTRD
Store	EVCSTORE	evc_store	EVTCLTST
End Event	EVCENDEVT	evc_endevt	EVTCLTEE
Set Option	EVCSETOPT	evc_setopt	EVTCLTSO

Table 6: Client APIs.

The sequence in which the APIs must be called is:

1. Start Event
2. Send Property
3. Send Data
4. Receive Response
5. Receive Data or Store
6. End Event

Of these, only the Start Event, Receive Response and End Event are required for processing an event. Set Option may be called at any time, but when the option will take effect depends on the current state of the API. Get Property may be called at any time, but will only return a non-blank value when called between Receive Response and End Event, except for the error properties which will be set if Set Option fails.

The string oriented APIs expect all text strings as zero-terminated (also known null-terminated, which means the last byte is hex 00 (last character is hex 0000 for DBCS)). The field oriented APIs contain additional parameters to specify the lengths of certain parameters. In addition these APIs pass a field for the return value. Fields which are only used by the field APIs are marked with a double asterisk (**) following.

It's important to note that all text parameters are variable length. Text parameters with no effective maximum length are denoted with a length of asterisk (*). For text parameters with a maximum length the length shown does not include the zero-terminator when the string APIs are used. That if the length is shown as 10, and you are using a string API you will actually have a field of 11 characters including the terminator. All maximum lengths are subject to change in future releases of Strategi.

All numeric parameters are four byte integers for the string APIs and decimal 15.5 for the field APIs. These are noted in the parameter lists by using the place-holder N.

All parameters lengths are followed by a single character 'I' or 'O', to indicate whether the parameter is an input or output parameter. No parameters are both input and output.

Start Event

API (field, string, OPM):
EVCSTREVT, evc_strevt, EVTCLTSE
Purpose:
Begins an event by contacting the event handler and establishing a two-way communications pipe with it.

Parameters:

Handler Name	10	I	Event handler's name, as set by the CRTEVTHDL command.
Handler Name Length**	N	I	Length of the Handler Name parameter.

Opcode	10	I	Opcode for processing the event.
Opcode Length**	N	I	Length of the Opcode parameter.
Success Flag**	1	O	Return flag; 1=success, 0=failure.

A given instance of the handler is dedicated to this client from when this API is called until an error condition or until End Event is called.

Send Property

API (field, string, OPM):
EVCSNDPRP, evc_sndprp, EVTCLTSP
Purpose:
Sends a property to the handler as part of the event. This API is called repeatedly, once for each property.

Parameters:

Keyword	*	I	Property keyword.
Keyword Length**	N	I	Length of the Keyword parameter.
Value	*	I	Property value.
Value Length**	N	I	Length of the Value parameter.
Success Flag**	1	O	Return flag; 1=success, 0=failure.

Properties beginning with an asterisk (*) are reserved for Strategi and will be rejected if used by customer code.

Send Data

API (field, string, OPM):
EVCSNDDTA, evc_snddta, EVTCLTSD
Purpose:
Sends a block of data to the handler as part of the event. This API is called repeatedly until all data is sent.

Parameters:

Data Block	*	I	Data to send.
Data Length	N	I	Length of the data in the Data Block parameter.
Success Flag**	1	O	Return flag; 1=success, 0=failure.

The API will send all of the data requested before returning. If it is unable to send the data, an error is indicated, and the event is aborted - the client should log the error and call End Event.

Receive Response

API (field, string, OPM):
EVCRCVRSP, evc_rcvrsp, EVTCLTRR
Purpose:
Waits for the handler to process the event data and begin transmitting the response.

Parameters:

Success Flag**	1	O	Return flag; 1=success, 0=failure.

Get Property

API (field, string, OPM):
EVCGETPRP, evc_getprp, EVTCLTGP

Purpose:
Retrieves a property value. Property keywords beginning with an asterisk (*) denote a property supplied by Strategi. All other properties are set by the handler.

Parameters:

Keyword	*	I	Name of the property.
Keyword Length**	N	I	Length of the property parameter.
Value	*	O	Variable to return the property value into.
Value Length	N	O	Maximum length of the value parameter. If the property value is longer than this, an error results.
Success Flag**	1	O	Return flag; 1=success, 0=failure.

If the current property value does not fit an error is generated. If the property is not defined (that is it was not transmitted by the handler) then a blank value is returned. This means that sending a blank property and not sending the property are logically equivalent, allowing a handler to avoid unnecessarily using communications bandwidth.

All Strategi client programs send the properties *SOURCE and *EVENT.

Special Properties	Description
*SYSTEMPROPERTYNAMES	A comma delimited list of all available system properties. These properties are resolved internally to the event API and available and valid for all events. These properties all begin with an asterisk (*).
*PROPERTYNAMES	A comma delimited list of all properties received from the handler program. These may include properties beginning with an asterisk when the handler is a Strategi program, but typically will not.
*ERRORCODE	The abstract high-level error code when an error has occurred, or blank to indicate no error. This should always be tested immediately before calling the end event API. Values can be: *NOHANDLER=Handler is not defined or not active, *DENIED=Client is denied access to the handler/opcode combination, *TIMEOUT=Handler appears to be active, but is not responding to event requests (the handler could be HELD, MSGW, etc, or it could be badly overloaded), or *ERROR=Some error occurred in using the event API.
*ERRORTEXT	A text description of the error, suitable for display to an end user, and in the format "SGInnnn Xxxx...". The Strategi message ID can be reliably interrogated for specific behavior, the text string following cannot be - it may change from release to release.
*ERRORDETAIL	An error specific detail message that should be logged or displayed for each error to assist in problem resolution.

Table 7: Client Special Properties.

Receive Data

API (field, string, OPM):
EVCRCVDTA, evc_rcvdta, EVTCLTRD
Purpose:

Receives a block of data from the event API. This API is called repeatedly until it receives a data length of -1, which indicates end of data.

Parameters:

Data Block	*	O	Variable to return the data into.
Data Maximum	N	I	Maximum length of the Data Block parameter.
Data Length	N	O	Pointer to a numeric entity to return the amount of data actually read.
Success Flag**	1	O	Return flag; 1=success, 0=failure.

The maximum number of bytes (not characters) put into data block is determined by the provided data maximum parameter. The number of bytes actually returned is returned in the data length parameter. For a receive CCSID which is a DBCS this represents half as many characters.

In CL and RPG all parameters are passed by reference, that is using pointers, so the term pointer may be unfamiliar to programmers of these languages. For these languages, a pointer parameter is simply a variable like any other in the program.

Store

API (field, string, OPM):
EVCSTORE, evc_store, EVTCLTST
Purpose:
Provides a convenient and easy way to store the event data to an IFS file.

Parameters:

File Name	*	I	Name of the IFS file to write the data to.
File Name Length**	N	I	Length of the File Name parameter.
Success Flag**	1	O	Return flag; 1=success, 0=failure.

If the file already exists it will be silently over-written.

End Event

API (field, string, OPM):
EVCENDEVT, evc_endevt, EVTCLTEE
Purpose:
Ends event processing and resets all variables, clears properties and prepares to start another event.

Parameters:
None

The End Event API can never fail.

This API is permitted at any time. If called while processing an event that is not in error; any pending data will be received from the handler. If the Receive Response API has not yet been called, then the communications to the handler will be abruptly terminated, causing an error condition in the handler. A correctly coded handler will reset for another event.

Set Option

API (field, string, OPM):
EVCSETOPT, evc_setopt, EVTCLTSO
Purpose:

Allows optional API behavior to be specified and the setting of individual preferences and requirements.

Parameters:

Keyword	*	I	Keyword of the option to set.
Keyword Length**	N	I	Length of the Keyword parameter.
Value	*	I	Value to set; exact content depends on keyword.
Value Length**	N	I	Length of the Value parameter.
Success Flag**	1	O	Return flag; 1=success, 0=failure.

The following table lists valid options and associated values.

Valid Options	Description/Values
CCSID	CCSID to use for all data. Values: Any pure CCSID, *UNICODE, *JOB, *CURRENT. *JOB indicates to use the CCSID of the job as it is when each translation is done; *CURRENT indicates to get the job CCSID as it currently is and set it as the CCSID. The default is the CCSID of the job when the event API is initialized.
TEXTCCSID	Set only the text (properties, opcode, etc) CCSID. Values and default as for CCSID.
SNDDTACCSID	Set only the CCSID for data that is sent. Default as for CCSID. Values: Any pure CCSID, *UNICODE, *JOB, *CURRENT, *HEX. *HEX indicates that no translation is done - the binary data is delivered exactly as sent.
RCVDTACCSID	Set only the CCSID for data that is received. Default as for CCSID. Values: Any pure CCSID, *UNICODE, *JOB, *CURRENT, *HEX. *HEX indicates that no translation is done - the binary data is delivered exactly as sent.
TIMEOUT	Set the communications send/receive timeout in seconds. Valid values are 1-2Gb in seconds. Default value is determined the Strategi value EVENTTIMEOUT which is issued at 300 seconds.
SNDTIMEOUT	Set the communications send timeout in seconds. See TIMEOUT.
RCVTIMEOUT	Set the communications receive timeout in seconds. See TIMEOUT.

Table 8: Client Valid Options and Values.

Event Handler Implementation

There are four essential steps to implementing an event handler: creating the executable program, registering the program to Strategi, adding a routing entry, and starting the event handler job.

Program Creation

There are two types of event handlers for the AS/400: *NATIVE and *JAVA. *NATIVE handlers become *PGM objects upon source compilation and reside in the QSYS.LIB file system. *JAVA handlers become CLASS objects upon source compilation, or class creation, and reside in the IFS.

Using the appropriate complier based upon source member type, create the executable object. For ILE field oriented languages such as RPG IV (i.e., source type is RPGLE), program creation is a two-step process, CRTRPGMOD followed by CRTPGM. Include the Strategi Binding Directory SGIBNDDIR on the Binding Directory parameter of the CRTPGM command so the compiler can locate the bindable Event API definitions.

Handler Registration

Registering the event handler to Strategi requires executing the CRTEVTHDL (Create Event Handler) command and completing the necessary parameters. Parameters on CRTEVTHDL define for Strategi job submission criteria such as program name, job description, performance monitoring, access control, etc.

Routing Entry Addition

Once the event handler is registered to Strategi, a routing entry, using the ADDEVTRTGE (Add Event Routing Entry) command, must be added to correlate an event to that handler and opcode.

It is necessary to assign an event name that corresponds to a particular handler and opcode because an event handler program is capable of processing multiple events/opcodes. This one-to-many relationship of the event handler to events provides additional layers of security for events. Remote client programs that want to use an event must refer to the event name and not the handler or opcode. This simplifies event invocation for the remote client by requiring a minimal amount of information in order to invoke the event.

The routing entry becomes a cross-reference between the event name and handler/opcode combination.

Handler Startup

Once the event handler program is created, the program is registered to Strategi, and a routing entry is made, the final step to activating an event is starting the event handler.

The STREVTHDL (Start Event Handler) command, either by command-line execution or selecting the appropriate option from the WRKEVTHDL (Work with Event Handlers) menu option, starts the event handler batch job in the Strategi subsystem. The job name will consist of the 8-character event handler name, as defined in the xxxEVTHDL commands, pre-pended with 'Y_'. For example, event handler MYEVT becomes job Y_MYEVT running in the Strategi subsystem.

Event Command Set

The following table lists the Event command set.

Command	Short Description
ADDEVTAUT	Add Event Authority
ADDEVTRTGE	Add Event Routing Entry
CHGEVTAUT	Change Event Authority
CHGEVTHDL	Change Event Handler
CHGEVTRTGE	Change Event Routing Entry
CHKEVTHDL	Check Event Handler
CLREVTPFR	Clear Event Performance Data
CRTEVTHDL	Create Event Handler
DLTEVTHDL	Delete Event Handler
DSPEVTAUT	Display Event Authority
DSPEVTHDL	Display Event Handler
DSPEVTPFR	Display Event Performance Data
DSPEVTRTGE	Display Event Routing Entry
ENDEVTHDL	End Event Handler
RMVEVTAUT	Remove Event Authority

RMVEVTPFR	Remove Event Performance Data
RMVEVTRTGE	Remove Event Routing Entry
RTVEVTAUT	Retrieve Event Authority
RTVEVTHDL	Retrieve Event Handler
RTVEVTPFR	Retrieve Event Performance Data
RTVEVTRTGE	Retrieve Event Routing Entry
STREVTHDL	Start Event Handler
WRKEVTAUT	Work With Event Authorities
WRKEVTHDL	Work With Event Handlers
WRKEVTPFR	Work With Event Performance Data
WRKEVTRTGE	Work With Event Routing Entries

Table 9: Event Command Set.

Event Handler Examples

The following examples demonstrate Event API use.

IFS Store Example

The following RPGLE event handler example stores event data into a file on the IFS of the AS/400. Seven of the eight Event Handler APIs are listed in this one example.

```
D*****************************************************************
D APIIND          S               1            * API RESULT INDICATOR
D
D RQSOPC          S              10            * REQUEST OPCODE
D RQSOLN          S              15P 5         * REQUEST OPCODE LENGTH
D RQSTMO          S              15P 5         * REQUEST TIMEOUT
D
D KWDDTA          S              20            * KEYWORD DATA
D KWDDLN          S              15P 5         * KEYWORD DATA LENGTH
D VALDTA          S             100            * VALUE DATA
D VALDLN          S              15P 5         * VALUE DATA LENGTH
D
D PRPNAM          S             100            * PROPERTY NAME
D PRPNLN          S              15P 5         * PROPERTY NAME LENGTH
D PRPVAL          S            1000            * PROPERTY VALUE
D PRPVMX          S              15P 5         * PROPERTY VALUE MAX LENGTH
D
D ERRCOD          S              10            * ERROR CODE
D ERRCMX          S              15P 5         * ERROR CODE MAX LENGTH
D ERRTXT          S             100            * ERROR TEXT
D ERRTMX          S              15P 5         * ERROR TEXT MAX LENGTH
D ERRDTL          S             100            * ERROR DETAIL
D ERRDMX          S              15P 5         * ERROR DETAIL MAX LENGTH
D
D MSGDTA          S              60            * MESSAGE DATA
D MSGDLN          S              15P 5         * MESSAGE DATA ACTUAL LENGTH
D
D STRFIL          S             100            * STORE FILENAME
D STRFMX          S              15P 5         * STORE FILENAME MAX LENGTH
D*****************************************************************
C*****************************************************************
C* MAIN LOOP
C*****************************************************************
C
C     *ENTRY        PLIST
C                   PARM                    PEVH            10
C
C                   EXSR      PGMINI
C
C     *INLR         DOWNE     '1'
C
C                   CLEAR                   RQSOPC
C
C                   CALLB     'EVHRCVEVT'
C                   PARM                    RQSOPC
C                   PARM      10            RQSOLN
C                   PARM      -1            RQSTMO
C                   PARM                    APIIND
C
C     RQSOPC        CASEQ     'IFSSTORE'    IFSSTORE
C     RQSOPC        CASEQ     '*STOP'       OPCSTOP
C                   CAS                     OPCERR
C                   END
C
C                   EXSR      CHKERROR
C
C                   CALLB     'EVHENDEVT'
C
C                   ENDDO
C
C                   EXSR      PGMEXI
C
C*****************************************************************
```

```
C* PRINCIPAL SUBROUTINES
C*****************************************************************
C* INI PROGRAM - INITIALIZE SUBROUTINE
C
C     PGMINI        BEGSR
C***
C* SETTING COMMUNICATION SND/RCV TIMEOUT TO 120 SECONDS.
C***
C                   CALLB     'EVHSETOPT'
C                   PARM      'TIMEOUT'     KWDDTA
C                   PARM      7             KWDDLN
C                   PARM      '120'         VALDTA
C                   PARM      3             VALDLN
C                   PARM                    APIIND
C
C                   ENDSR
C
C*****************************************************************
C* CHECK API ERROR - CHECK FOR API ERRORS
C
C     CHKERROR      BEGSR
C
C                   CALLB     'EVHGETPRP'
C                   PARM      '*ERRORCODE'  PRPNAM
C                   PARM      10            PRPNLN
C                   PARM      ' '           ERRCOD
C                   PARM      10            ERRCMX
C                   PARM                    APIIND
C
C     ERRCOD        IFNE      *BLANKS
C
C                   CALLB     'EVHGETPRP'
C                   PARM      '*ERRORTEXT'  PRPNAM
C                   PARM      10            PRPNLN
C                   PARM      ' '           ERRTXT
C                   PARM      100           ERRTMX
C                   PARM                    APIIND
C
C                   CALLB     'EVHGETPRP'
C                   PARM      '*ERRORDETAIL'PRPNAM
C                   PARM      12            PRPNLN
C                   PARM      ' '           ERRDTL
C                   PARM      100           ERRDMX
C                   PARM                    APIIND
C                   ENDIF
C***
C* RECOMMEND OUTPUTTING MESSAGE TO JOBLOG OR ANOTHER
C*  MECHANISM.
C***
C                   ENDSR
C
C*****************************************************************
C* END PROGRAM - PERFORM FINAL CLEANUP BEFORE EXITING PGM
C
C     PGMEXI        BEGSR
C
C                   ENDSR
C
C*****************************************************************
C* RESERVED OPCODES
C*****************************************************************
C* STOP REQUEST - SHUTDOWN EVENT HANDLER
C
C     OPCSTOP       BEGSR
C
C                   EVAL      *INLR = '1'
C
C                   ENDSR
C
C*****************************************************************
```

```
C* OPCODE ERROR – UNKNOWN OPCODE PROCESSING
C
C     OPCERR        BEGSR
C
C                   ENDSR
C
C*****************************************************************
C* USER DEFINED OPCODES
C*****************************************************************
C
C*****************************************************************
C* IFSSTORE – STORE EVENT CONTENT TO IFS FILE /tmp/event123.txt
C
C     IFSSTORE      BEGSR
C***
C* SETTING RECEIVE DATA CCSID TO CODE PAGE 819.
C***
C                   CALLB     'EVHSETOPT'
C                   PARM      'RCVDTACCSID' KWDDTA
C                   PARM      11            KWDDLN
C                   PARM      '819'         VALDTA
C                   PARM      3             VALDLN
C                   PARM                    APIIND
C
C                   EVAL      STRFIL = '/tmp/event123.txt'
C
C                   CALLB     'EVHSTORE'
C                   PARM                    STRFIL
C                   PARM      17            STRFMX
C                   PARM                    APIIND
C***
C* RESETTING RECEIVE DATA CCSID TO THE JOB'S CURRENT
C*  CCSID VALUE.
C***
C                   CALLB     'EVHSETOPT'
C                   PARM      'RCVDTACCSID' KWDDTA
C                   PARM      11            KWDDLN
C                   PARM      '*CURRENT'    VALDTA
C                   PARM      8             VALDLN
C                   PARM                    APIIND
C***
C* SEND THE RESPONSE MESSAGE AS AN XML DOCUMENT.
C***
C                   EVAL      PRPNAM = 'header.content-type'
C                   EVAL      PRPVAL = 'text/xml; charset=utf-8'
C
C                   CALLB     'EVHSNDPRP'
C                   PARM                    PRPNAM
C                   PARM      30            PRPNLN
C                   PARM                    PRPVAL
C                   PARM      30            PRPVMX
C                   PARM                    APIIND
C
C                   CALLB     'EVHSNDDTA'
C                   PARM      'SUCCESS'     MSGDTA
C                   PARM      7             MSGDLN
C                   PARM                    APIIND
C
C                   ENDSR
C
C*****************  END  OF  EXAMPLE  ******************************
```

Receive Data Example

The following RPGLE event handler example receives event data using the EVHRCVDTA API. The data is received in 60 byte blocks, thus allowing for variable length data size.

```
D*****************************************************************
D APIIND          S              1            * API RESULT INDICATOR
D
D RQSOPC          S             10            * REQUEST OPCODE
D RQSOLN          S             15P 5         * REQUEST OPCODE LENGTH
D RQSTMO          S             15P 5         * REQUEST TIMEOUT
D
D KWDDTA          S             20            * KEYWORD DATA
D KWDDLN          S             15P 5         * KEYWORD DATA LENGTH
D VALDTA          S            100            * VALUE DATA
D VALDLN          S             15P 5         * VALUE DATA LENGTH
D
D PRPNAM          S            100            * PROPERTY NAME
D PRPNLN          S             15P 5         * PROPERTY NAME LENGTH
D PRPVAL          S           1000            * PROPERTY VALUE
D PRPVMX          S             15P 5         * PROPERTY VALUE MAX LENGTH
D
D ERRCOD          S             10            * ERROR CODE
D ERRCMX          S             15P 5         * ERROR CODE MAX LENGTH
D ERRTXT          S            100            * ERROR TEXT
D ERRTMX          S             15P 5         * ERROR TEXT MAX LENGTH
D ERRDTL          S            100            * ERROR DETAIL
D ERRDMX          S             15P 5         * ERROR DETAIL MAX LENGTH
D
D MSGDTA          S             60            * MESSAGE DATA
D MSGDMX          S             15P 5         * MESSAGE DATA MAX LENGTH
D MSGDLN          S             15P 5         * MESSAGE DATA ACTUAL LENGTH
D RCVCNT          S              2P 0         * RECEIVE COUNT
D*****************************************************************
C*****************************************************************
C* MAIN LOOP
C*****************************************************************
C
C     *ENTRY        PLIST
C                   PARM                    PEVH            10
C
C                   EXSR      PGMINI
C
C     *INLR         DOWNE     '1'
C
C                   CLEAR                   RQSOPC
C
C                   CALLB     'EVHRCVEVT'
C                   PARM                    RQSOPC
C                   PARM      10            RQSOLN
C                   PARM      -1            RQSTMO
C                   PARM                    APIIND
C
C     RQSOPC        CASEQ     'RCVDATA'     RCVDATA
C     RQSOPC        CASEQ     '*STOP'       OPCSTOP
C                   CAS                     OPCERR
C                   END
C
C                   EXSR      CHKERROR
C
C                   CALLB     'EVHENDEVT'
C
C                   ENDDO
C
C                   EXSR      PGMEXI
C
C*****************************************************************
```

```
C* PRINCIPAL SUBROUTINES
C*****************************************************************
C* INI PROGRAM – INITIALIZATION SUBROUTINE
C
C     PGMINI        BEGSR
C***
C* SETTING COMMUNICATION SND/RCV TIMEOUT TO 120 SECONDS.
C***
C                   CALLB     'EVHSETOPT'
C                   PARM      'TIMEOUT'     KWDDTA
C                   PARM      7             KWDDLN
C                   PARM      '120'         VALDTA
C                   PARM      3             VALDLN
C                   PARM                    APIIND
C
C                   ENDSR
C
C*****************************************************************
C* CHECK API ERROR – CHECK FOR API ERRORS
C
C     CHKERROR      BEGSR
C
C                   CALLB     'EVHGETPRP'
C                   PARM      '*ERRORCODE'  PRPNAM
C                   PARM      10            PRPNLN
C                   PARM      ' '           ERRCOD
C                   PARM      10            ERRCMX
C                   PARM                    APIIND
C
C     ERRCOD        IFNE      *BLANKS
C
C                   CALLB     'EVHGETPRP'
C                   PARM      '*ERRORTEXT'  PRPNAM
C                   PARM      10            PRPNLN
C                   PARM      ' '           ERRTXT
C                   PARM      100           ERRTMX
C                   PARM                    APIIND
C
C                   CALLB     'EVHGETPRP'
C                   PARM      '*ERRORDETAIL'PRPNAM
C                   PARM      12            PRPNLN
C                   PARM      ' '           ERRDTL
C                   PARM      100           ERRDMX
C                   PARM                    APIIND
C                   ENDIF
C***
C* RECOMMEND OUTPUTTING MESSAGE TO JOBLOG OR ANOTHER
C*  MECHANISM.
C***
C                   ENDSR
C*****************************************************************
C* END PROGRAM – PERFORM FINAL CLEANUP BEFORE EXITING PGM
C
C     PGMEXI        BEGSR
C
C                   ENDSR
C
C*****************************************************************
C* RESERVED OPCODES
C*****************************************************************
C* STOP REQUEST – SHUTDOWN EVENT HANDLER
C
C     OPCSTOP       BEGSR
C
C                   EVAL      *INLR = '1'
C
C                   ENDSR
C
C*****************************************************************
```

```
C* OPCODE ERROR – UNKNOWN OPCODE PROCESSING
C
C     OPCERR        BEGSR
C
C                   ENDSR
C
C*****************************************************************
C* USER DEFINED OPCODES
C*****************************************************************
C
C*****************************************************************
C* RCVDATA – RECEIVE EVENT DATA USING THE EVHRCVDTA API.
C
C     RCVDATA       BEGSR
C***
C* GET TWO HTTP HEADER PROPERTIES VALUES. THIS EXAMPLE DOES
C*  NOT UTILIZE THE PROPERTY VALUE, ONLY DEMONSTRATES HOW TO
C*  USE THE EVHGETPRP API TO OBTAIN IT.
C***
C                   EVAL      PRPNAM = 'header.content-length'
C
C                   CALLB     'EVHGETPRP'
C                   PARM                PRPNAM
C                   PARM      50        PRPNLN
C                   PARM                PRPVAL
C                   PARM      1000      PRPVMX
C                   PARM                APIIND
C
C                   EVAL      PRPNAM = 'header.content-type'
C
C                   CALLB     'EVHGETPRP'
C                   PARM                PRPNAM
C                   PARM      50        PRPNLN
C                   PARM                PRPVAL
C                   PARM      1000      PRPVMX
C                   PARM                APIIND
C
C                   Z-ADD     0         RCVCNT
C                   Z-ADD     1         MSGDLN
C***
C* RECEIVE THE DATA IN BLOCKS BY LOOPING UNTIL ALL
C*  DATA HAS BEEN RECEIVED, INDICATED BY MSGDLN=-1. AFTER
C*  EACH DATA BLOCK IS RECEIVED, A CUSTOMER-WRITTEN
C*  PROGRAM IS CALLED TO PROCESS THE DATA BLOCK.
C* THIS EXAMPLE ASSUMES ALL DATA WILL BE RECEIVED WITHIN
C*  10 ITERATIONS THROUGH THE LOOP OR STOP RECEIVING DATA.
C***
C     MSGDLN        DOWNE     -1
C     RCVCNT        ADD       1         RCVCNT
C                   CLEAR               MSGDTA
C                   CALLB     'EVHRCVDTA'
C                   PARM                MSGDTA
C                   PARM      60        MSGDMX
C                   PARM                MSGDLN
C                   PARM                APIIND
C     APIIND        IFEQ      '0'
C                   EXSR      CHKERROR
C                   EXSR      OPCERR
C                   GOTO      RCVDATAE
C                   ENDIF
C     RCVCNT        IFGT      10
C                   Z-ADD     -1        MSGDLN
C                   ENDIF
C                   CALL      'PRCDATA'
C                   PARM                MSGDTA
C                   ENDDO
C
C                   CALL      'PRCDATA'
C                   PARM      '--DONE--'    MSGDTA
C     RCVDATAE      ENDSR
C*****************  END  OF  EXAMPLE  ******************************
```

Invoking Strategi Events in a URL

Once you have you configured your Event Handler and your Event Routing Entry, you can begin testing.

HTTP Source

The Event can be invoked on an HTML page, as is shown in the following example:

```
http://www.businesslink.com/doc/testdoc.htm;*EVENT=TESTEVENT
```

Where "testdoc.htm" is the document on which you wish to invoke the Event and "TESTEVENT" is the name of the Event Routing Entry to use.

*Note: Your Event Routing Entry needs to be configured with an Event Source of either *ANY or *HTTP.*

The semi-colon is required because you do not want to embed *EVENT in the URL path, as it will affect how the browser interprets relative links. The HTML document may or may not have any relevance. It is completely dependent on what the event handler does. The requested document is an available property to use.

Parameters may be passed in the URL. These parameters will be available as properties to the Event. For example:

```
http://www.businesslink.com/doc/testdoc.htm;*EVENT=TESTEVENT?PARM1
=VAL1&PARM2=VAL2
```

The Event Handler will use EVHGETPRP to get the URL data properties (parameters), specifying the keywords as `"urldata.PARM1"` and `"urldata.PARM2"` to get their values.

Form elements are also available to the Event Handler as properties and are considered URL data.

Event Handler Log Files

When an event handler is started a corresponding log file is created in the Strategi IFS directory '/STRATEGI/tmp'. This log file has the following naming convention:

```
/STRATEGI/tmp/handler.EVTSER-744652-F6E0CFF6-001
```

Where

- "EVTSER" is the name of the event handler
- "744652" is the corresponding OS/400 job number (the corresponding job runs in the Strategi subsystem)
- "F6E0CFF6" is a randomly generated number for the server instance
- "001" is the log file sequence

List of Tables

List of Figures

Index