

Entire Event Management API

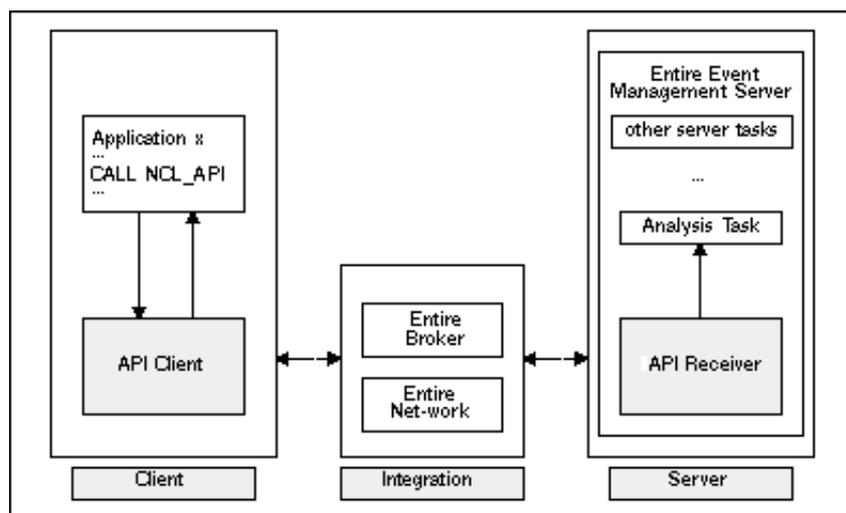
This section covers the following topics:

- Introduction
- API for Natural - platforms OS/390, VSE/ESA and BS2000/OSD
- API for C - platform Windows 3.x

Introduction

The Entire Event Management API enables applications to forward exception messages, so-called events, to the Entire Event Management server for further analysis. The server determines by means of filter and automation rule definitions provided by the administrator whether the event must be logged in the database and which automated actions must be executed.

The API functions are designed as client/server functions. This means that each function can be split into a client part and a server part. The following figure illustrates this principle of operation:



The client part of the API is provided on the mainframe platforms OS/390, VSE/ESA and BS2000/OSD with a Natural CALLNAT interface and on the Windows 3.x platform with a C language call interface.

The server part, called **API Receiver**, is provided on the mainframe platforms only. It maps the API event message format to an internal message format, which is then forwarded to the Analysis Task of the Entire Event Management Server via the Entire System Server view EVENTING. The API Receiver registers as a service with Entire Broker. It can run as a subtask of the Entire Event Management Server or as a separate batch job which does not need to run on the same network node as the Entire Event Management Server.

If the client application is located on the same network node as the Entire Event Management Server, it can be specified in a configuration file that the API Receiver service is local for the client. In this case, the integration mechanism provided with Entire Broker / Entire Net-work is not involved in the communication. The service which forwards the event message to the Analysis Task via EVENTING view is then called directly by the API client, thus reducing the communication overhead.

API for Natural - platforms OS/390, VSE/ESA and BS2000/OSD

Call Format

```
CALLNAT 'NCL_API'
  USING FUNCTION
    TARGET_SERVICE
    BUFFER
    ERROR_INFO
    RETURN_CODE
```

where (see also member DOCPI00A in library SYSNCLPI):

Name	Format	Usage
FUNCTION	I2	Input, mandatory. An integer field whose value relates to an API function.
TARGET_SERVICE	A16	Input, mandatory. The name of the API Receiver service. This name must be registered in a configuration file (see Configuring the API Receiver).
BUFFER	A1(1:4096)	Input, mandatory. An array which contains a structure specific to the function.
ERROR_INFO	A1(1:200)	Output. A structure which contains error information provided by deeper call levels.
RETURN_CODE	I2	Output. An integer field containing a return code (see Return Codes for the format and content).

API Functions

The following functions are currently supported (defined in LDA NCLPI--E, see also member DOCPI--E in library SYSNCLPI):

Function	Value	Description
NCL_FC_EVFORWARD	1	Forward an event message to the Entire Event Management Server.
NCL_FC_STOPSERVICE	9999	Stop the API Receiver service.

NCL_FC_EVFORWARD - 1

Use this function to forward an event message to the Entire Event Management Server. Provide values for the parameters of the following data structure (defined in LDA NCLPI01L, see also member DOCPI01L in library SYSNCLPI) and put this structure into the BUFFER parameter:

Name	Format	Usage
EV_MSGID	A10	Input, mandatory. This attribute identifies the event message.
EV_TEXT	A180	Input, optional. The message text. This attribute can be empty especially when a language-dependent representation is required and dynamic substitute strings are provided with EV_TEXT_VAR1 .. 5.
EV_CATEGORY	A32	Input, optional. This attribute can be used as classification criterion.
EV_SEVERITY	A1	Input, optional. This attribute can be used to indicate the severity of the event.
EV_SOURCE_NODE	A32	Input, optional. The originator network node where the event occurred.
EV_SOURCE_APPL	A32	Input, optional. The originator application which reports the event.
EV_JOBNAME	A8	Input, optional. The name of the originating job which reports the event
EV_JOBID	A8	Input, optional. The identifier of the originating job which reports the event.
EV_TEXT_VAR1	A64	Input, optional. A text string which is to be dynamically replaced in the language-dependent message text during representation.
EV_TEXT_VAR2	A64	Input, optional. A text string which is to be dynamically replaced in the language-dependent message text during representation.
EV_TEXT_VAR3	A64	Input, optional. A text string which is to be dynamically replaced in the language-dependent message text during representation.
EV_TEXT_VAR4	A64	Input, optional. A text string which is to be dynamically replaced in the language-dependent message text during representation.
EV_TEXT_VAR5	A64	Input, optional. A text string which is to be dynamically replaced in the language-dependent message text during representation.

NCL_FC_STOPSERVICE - 9999

Use this function to stop the API Receiver service. Be sure that there are no other clients which still need the API Receiver service. To perform this function successfully, you must fill the BUFFER parameter with the constant value NCL_INTERNAL_CALL (see also members DOCPI00A and DOCPI--E in library SYSNCLPI).

Return Codes

The following return codes are sent back to the API caller (defined in LDA NCLPI--E, see also member DOCPI--E in library SYSNCLPI):

0 NCL_RT_NORMAL

Expl.:	Successful execution.
Actn.:	None.

1 NCL_RT_IVFUNC

Expl.:	Invalid function.
Actn.:	Provide a valid function in the parameter FUNCTION (see DOCPI--E in library SYSNCLPI for valid values).

2 NCL_RT_IVSERVICE

Expl.:	Invalid service.
Actn.:	Provide a valid service name in the parameter TARGET_SERVICE. This name must be registered in the configuration file (see the subsection Configuring the API Receiver).

3 NCL_RT_IVESYNODE

Expl.:	Invalid Entire System Server node.
Actn.:	Provide a valid value for the configuration parameter ESY_Node in the configuration file (see the subsection Configuring the API Receiver).

4 NCL_RT_RUNERR

Expl.:	A runtime error has occurred. More detailed information (for example, original error code/text, erroneous program and line of program, reporting component) is provided in the structure ERROR_INFO (see member DOCPI00A in library SYNCLPI for an explanation).
Actn.:	Analyze the content of ERROR_INFO and act accordingly. Possible values for ERROR_CLASS: <ul style="list-style-type: none"> ● N = Natural runtime error. ● P = a runtime error reported by Entire System Server. ● S = a runtime error reported by the SAT component.

5 NCL_RT_COMMERR

Expl.:	An error has occurred in the communication infrastructure. More detailed information (for example, original error code/text, erroneous program and line of program, reporting component) is provided in the structure ERROR_INFO (see member DOCPI00A in library SYNCLPI for an explanation).
Actn.:	Analyze the content of ERROR_INFO and act accordingly. Possible values for ERROR_CLASS: <ul style="list-style-type: none"> ● S = a communication error reported by the SAT component.

6 NCL_RT_BACKERR

Expl.:	An error has occurred in the API back-end. More detailed information (for example, original error code/text, erroneous program and line of program, reporting component) is provided in the structure ERROR_INFO (see member DOCPI00A in library SYNCLPI for an explanation).
Actn.:	Analyze the content of ERROR_INFO and act accordingly. Possible values for ERROR_CLASS: <ul style="list-style-type: none"> ● I an internal error detected by a back-end program ● U a user error detected by a back-end program.

7 NCL_RT_MAXCONV_EXCEEDED

Expl.:	The maximum number of client conversations which the API Receiver service can handle in parallel (currently 10) has been exceeded.
Actn.:	Retry the client request after a short wait.

8 NCL_RT_ALIENREQ

Expl.:	The client has provided a value for the parameter TARGET_SERVICE which is different from the name under which the API Receiver service was started.
Actn.:	Be sure to use identical names for referring to the API Receiver service in both the client and the server environment (see also the subsections Configuring the API Receiver and Starting the API Receiver).

99 NCL_RT_SERVICE_STOPPED

Expl.:	This return code informs the client that the API Receiver service has been stopped because of its previous NCL_FC_STOPSERVICE request.
Actn.:	None.

100 NCL_RT_IVVERSION

Expl.:	This code is returned by the back-end or server part of the API and means that the version of the client part is not compatible with the version of the server part.
Actn.:	Make sure that the installed Entire Event Management versions of the client and the server part are always compatible.

101 NCL_RT_IVMSGID

Expl.:	The content of the attribute EV_MSGID in the parameter BUFFER is not valid.
Actn.:	Provide a non-blank value for the attribute EV_MSGID.

Examples

The program XAPI01-P in library SYSNCLPI provides an example of how to use the Entire Event Management API.

Configuring the API Receiver

The API Receiver is a service which registers with Entire Broker. Entire Broker identifies this service by the attributes CLASS, SERVER and SERVICE. The server must pass these attributes to Entire Broker to REGISTER requests. The client must pass these attributes to Entire Broker to SEND requests.

The SAT component facilitates the addressing and configuration of services by allowing the service to be addressed with a symbolic name, which is registered in a text member of library SYSSATU (see also subsection SAT in Client/Server Environments in Section Installing System Automation Tools) of the SAT Installation and Customization Documentation. The syntax for the symbolic name is as follows:

[<member-name> .] <section-name> .

For example, the name: **NCLPARMS.ncl_api** addresses section **ncl_api** in member **NCLPARMS** of library **SYSSATU**. If the <member-name> token is omitted, it is assumed that the section is located in the text member **SATSRV**.

SATSRV Configuration Parameters

Name	Description
TYPE	Type of communication. Must currently be ACI.
SERVER-CLASS	Corresponds to the parameter CLASS in the SDPA structure.
SERVER-NAME	Corresponds to the parameter SERVER in the SDPA structure.
SERVICE	Corresponds to the parameter SERVICE in the SDPA structure.
USER-ID	Corresponds to the parameter UID in the SDPA structure.
WAIT-TIME	Corresponds to the parameter WAIT in the SDPA structure.
Trace	This toggle can have the values on or off . If set to on , trace messages are produced for better diagnostics.
Service_Location	Indicates whether the API Receiver service is located locally (value l) or remotely (value r) to the client. If it is located locally, the back-end modules are called directly from the client part of the API, and the Entire Broker communication infrastructure is not involved.
Local_Node	The value provided here identifies the network node where the client runs and is taken as default for the event message attribute EV_SOURCE_NODE.
ESY_Node	Node number of the Entire System Server nucleus to be used by the APIReceiver service.

Example: SATSRV Parameters

```
ncl_api SATSRV TYPE=ACI
                BROKER-ID=BKR034
                SERVER-CLASS=NCL
                SERVER-NAME=IBM1
                SERVICE=EventReceiver
                USER-ID=huhu
                WAIT-TIME=30S
                Trace=on
                Service_Location=r
                Local_Node=ibm1
                ESY_Node=114
```

Starting the API Receiver

The API Receiver service can be started in two different ways:

- as **subtask** of the Entire Event Management Server.

In this case, the parameter API Receiver Service of the Miscellaneous Server Parameters group must contain the name of the service as registered in the library SYSSATU. The service is then started automatically during startup of Entire Event Management Server, and its status can be checked with the **Server Statistic Monitor**.

- as a separate **batch job**.

In this case, the service does not necessarily run on the same network node as the Entire Event Management Server. The member E-PIRCVR in the NCLnnn.SRCE library provides an sample JCL skeleton.

Make sure that the LFILE assignment for Entire Event Management System File 2 (LFILE number 202) corresponds to the SERVSYSF parameter of the Entire Event Management Server to which the API Receiver service will forward the event message, because SERVSYSF is used to uniquely address the correct EVENTING message queue.

API for C - platform Windows 3.x

ncl_api - NCL_API

Send client requests to the API Receiver service.

Synopsis

```
#include <ncl_api.h>
void ncl_api (NCL_API *arg);
typedef struct NCL_API
{
    long int    function;
    unsigned char target_service[16];
    unsigned char buffer[4096];
    unsigned char error_info[200];
    long int    return_code;
} NCL_API
```

Description

The client request specified in the **buffer** argument according to the desired **function** is sent to the API Receiver service addressed by the **target_service** argument.

Arguments

Argument	Description
function	Input, mandatory. An integer field whose value relates to an API function.
target_service	Input, mandatory. Specifies The name of the API Receiver service. This name must be registered in a configuration file (see Configuring the API Client).
buffer	Input, mandatory. Contains a structure specific to the function.
error_info	Output. A structure which contains error information provided by deeper call levels.
return_code	Output. An integer field containing a return code (see Return Codes for the format and content).

API Functions

The following function codes are currently supported (defined in header file **ncl_api.h**):

Function	Value	Description
NCL_FC_EVFORWARD	1	Forward an event message to the Entire Event Management Server.
NCL_FC_STOPSERVICE	9999	Stop the API Receiver service.

They correspond to the following functions which are also defined as prototypes in **ncl_api.h**.

ncl_forward_event - NCL_API01

Forward an event message to the Entire Event Management Server.

Synopsis

```
#include <ncl_api.h>
void ncl_forward_event (NCL_API01 *arg);
typedef struct NCL_API01
{
    unsigned char target_service[16];
    unsigned char buffer[4096];
    unsigned char error_info[200];
    long int return_code;
} NCL_API01
```

Description

Use this function to forward an event message to the Entire Event Management Server. Provide values for the items of the following data structure (defined in header file **ncl_api01.h**) and put this structure into the **buffer** argument:

```
typedef struct NCL_EVENT
{
    unsigned char ev_msgid [10];
    unsigned char ev_text [180];
    unsigned char ev_category [32];
    unsigned char ev_severity;
    unsigned char ev_source_node [32];
    unsigned char ev_source_appl [32];
    unsigned char ev_jobname [8];
    unsigned char ev_jobid [8];
    unsigned char ev_text_var1[64];
    unsigned char ev_text_var2[64];
    unsigned char ev_text_var3[64];
    unsigned char ev_text_var4[64];
    unsigned char ev_text_var5[64];
} NCL_EVENT
```

Arguments

Argument	Description
ev_msgid	Input, mandatory. This attribute identifies the event message.
ev_text	Input, optional. The message text. This attribute can be empty especially when language-dependent representation is required and dynamic substitute strings are provided with ev_text_var1.. 5 .
ev_category	Input, optional. This attribute can be used as classification criterion.
ev_severity	Input, optional. This attribute can be used to indicate the severity of the event.
ev_source_node	Input, optional. The originator network node where the event occurred.
ev_source_appl	Input, optional. The originator application which reports the event.
ev_jobname	Input, optional. The name of the originating job which reports the event.
ev_jobid	Input, optional. The identifier of the originating job which reports the event
ev_text_var1.. 5	Input, optional. A text string which is to be dynamically replaced in the language-dependent message text during representation.

ncl_stop_service - NCL_API01

Stop the API Receiver service.

Synopsis

```
#include <ncl_api.h>
void ncl_stop_service (NCL_API01 *arg);
typedef struct NCL_API01
{
    unsigned char target_service[16];
    unsigned char buffer[4096];
    unsigned char error_info[200];
    long int return_code;
}
NCL_API01
```

Description

Use this function to stop the API Receiver service. Be sure that there are no other clients which still need the API Receiver service. To perform this function successfully, you must fill the **buffer** argument with the constant value NCL_INTERNAL_CALL (defined in header file **ncl_api.h**).

Return Codes

The following return codes are sent back to the API caller. They are defined in header file **ncl_api.h**.

0 NCL_RT_NORMAL

Expl.:	Successful execution.
Actn.:	None.

1 NCL_RT_IVFUNC

Actn.:	Provide a valid function in the argument function .
--------	--

2 NCL_RT_IVSERVICE

Actn.:	Provide a valid service name in the argument target_service . This name must be registered in the configuration file (see the subsection Configuring the API Client).
--------	--

3 NCL_RT_IVESYNODE

Expl.:	Invalid Entire System Server node.
Actn.:	Provide a valid value for the configuration parameter ESY_Node in the configuration file of the API Receiver environment (see the subsection Configuring the API Receiver).

4 NCL_RT_RUNERR

Expl.:	A runtime error has occurred. More detailed information (for example, original error code/text, erroneous program and line of program, reporting component) is provided in the structure error_info (see header file ncl_api.h for an explanation).
Actn.:	Analyze the content of error_info and act accordingly. Possible values for error_class : <ul style="list-style-type: none"> ● N = Natural runtime error. ● P = a runtime error reported by Entire System Server. ● S = a runtime error reported by the SAT component.

5 NCL_RT_COMMERR

Expl.:	An error has occurred in the communication infrastructure. More detailed information (for example, original error code/text, erroneous program and line of program, reporting component) is provided in the structure error_info (see header file ncl_api.h for an explanation).
Actn.:	Analyze the content of error_info and act accordingly. Possible values for error_class : <ul style="list-style-type: none"> ● S a communication error reported by the SAT component.

6 NCL_RT_BACKERR

Expl.:	An error has occurred in the API back-end. More detailed information (for example, original error code/text, erroneous program and line of program, reporting component) is provided in the structure error_info (see header file ncl_api.h for an explanation).
Actn.:	Analyze the content of error_info and act accordingly. Possible values for error_class : <ul style="list-style-type: none"> ● I an internal error detected by a back-end program. ● U a user error detected by a back-end program.

7 NCL_RT_MAXCONV_EXCEEDED

Expl.:	The maximum number of client conversations which the API Receiver service can handle in parallel (currently 10) has been exceeded.
Actn.:	Retry the client request after a short wait.

8 NCL_RT_ALIENREQ

Expl.:	The client has provided a value for the argument target_service which is different from the name under which the API Receiver service was started.
Actn.:	Be sure to use identical names for referring to the API Receiver service in both the client and the server environment (see also the subsections Configuring the API Receiver and Starting the API Receiver).

99 NCL_RT_SERVICE_STOPPED

Expl.:	This return code informs the client that the API Receiver service has been stopped because of its previous NCL_FC_STOPSERVICE request.
Actn.:	None.

100 NCL_RT_IVERSION

Expl.:	This code is returned by the back-end or server part of the API and means that the version of the client part is not compatible with the version of the server part.
Actn.:	Make sure that the installed Entire Event Management versions of the client and the server part are always compatible.

101 NCL_RT_IVMSGID

Expl.:	The content of the attribute ev_msgid in the buffer argument is not valid.
Actn.:	Provide a non-blank value for the attribute ev_msgid .

Examples

The program **xapi01.c**, contained on the installation diskette, provides an example of how to use the Entire Event Management API. The executable program **xapi01.exe** can be invoked as follows (synopsis):

```
xapi01 -f <function> -s<target_service> -i<ev_msgid> -t<ev_text>
```

where **function**, **target_service**, **ev_msgid** and **ev_text** correspond to the arguments and attributes described above.

Configuring the API Client

On the API client site, the way in which the API Receiver service is addressed and used must be customized. The API Receiver is a service which registers with Entire Broker. Entire Broker identifies this service by the attributes CLASS, SERVER and SERVICE. The server must pass these attributes to Entire Broker to REGISTER requests. The client must pass these attributes to Entire Broker to SEND requests.

The SAT component facilitates the addressing and configuration of services by allowing the service to be addressed with a symbolic name, which refers to a section defined in the **sat.ini** file located in the SAT directory on the API client site.

The following configuration parameters can be specified in the **sat.ini** file:

Name	Description
Type	Type of communication. Must currently be ACI.
BrokerID	Corresponds to the parameter BROKER-ID in the Entire Broker attribute file and SDPA structure.
Class	Corresponds to the parameter CLASS in the SDPA structure.
Server	Corresponds to the parameter SERVER in the SDPA structure.
Service	Corresponds to the parameter SERVICE in the SDPA structure.
UserID	Corresponds to the parameter UID in the SDPA structure.
WaitTime	Corresponds to the parameter WAIT in the SDPA structure.
Trace	This toggle can have the values on or off . If set to on , trace messages are produced for better diagnostics.
LocalNode	The value provided here identifies the network node where the client runs and is taken as default for the event message attribute ev_source_node .

Example: sat.ini Parameters for API Receiver

```
[ncl_api]
Type=ACI
BrokerID=BKR034
Class=NCL
Server=IBM1
Service=EventReceiver
UserID=huhu
WaitTime=30S
Trace=on
LocalNode=pchka
```