Configuring Broker for Internationalization

It is assumed that you have read the document *Internationalization with EntireX* and are familiar with the various internationalization approaches described there.

This chapter explains in detail how to configure the broker for the various internationalization approaches, how to write a translation user exit and how to write a SAGTRPC user exit. It covers the following topics:

- Configuring Translation
- Configuring Translation User Exits
- Configuring ICU Conversion
- Configuring SAGTRPC User Exits
- Writing Translation User Exits
- Writing SAGTRPC User Exits
- Building and Installing ICU Custom Converters

See also *What is the Best Internationalization Approach to use?*

Configuring Translation

To configure translation

- In the Broker attribute file, set the service-specific or topic-specific broker attribute `TRANSLATION` to `SAGTCHA` as the name of the translation routine. Example:

  ```
  TRANSLATION=SAGTCHA
  ```

Configuring Translation User Exits

To configure translation user exits

As a prerequisite, the user-written translation routine DLL must be accessible to the Broker worker threads.

1. Copy the user-written translation routine DLL into the EntireX *bin* folder.
2. In the Broker attribute file, set the service-specific or topic-specific broker attribute `TRANSLATION` to the name of the user-written translation routine. Example:

   ```
   TRANSLATION=MYTRANS
   ```

   or

1. Place the user-written translation routine DLL in a folder of your choice. Spaces in the path name are not allowed.
2. In the Broker attribute file, set the service-specific or topic-specific broker attribute `TRANSLATION` to the full path name of the folder of the user-written translation routine. Example:

   ```
   TRANSLATION="C:\MyDir\MyTrans\MyTrans.dll"
   ```

Configuring ICU Conversion

To configure ICU conversion

1. In the Broker attribute file, set the service-specific attribute `CONVERSION`. Examples:

   - ICU Conversion with SAGTCHA for *ACI-based Programming*:

     ```
     CONVERSION=(SAGTCHA,TRACE=1,OPTION=SUBSTITUTE)
     ```

   - ICU Conversion with SAGTRPC for *RPC-based Components* and *Reliable RPC*:

     ```
     CONVERSION=(SAGTRPC,TRACE=2,OPTION=STOP)
     ```

   We recommend always using SAGTRPC for RPC data streams. *Conversion with Multibyte, Double-byte and other Complex Codepages* will always be correct, and *Conversion with Single-byte Codepages* is also efficient because SAGTRPC detects single-byte codepages automatically. See *Conversion Details*.

2. Optionally configure a `CONVERSION OPTION` to tune error behavior to meet your requirements; see *`OPTION` Values for Conversion*.
3. For the Broker attribute, check if ICU conversion is possible, that is, the attribute `ICU-CONVERSION` is either

   - not defined, its default is `YES`

- set to YES

To configure locale string defaults (optional)

- If the broker's locale string defaults do not match your requirements (see *Broker's Locale String Defaults*), we recommend you assign suitable locale string defaults for your country and region, see the respective attribute in *Codepage-specific Attributes* under *Broker Attributes* for how to customize the broker's locale string defaults.

To customize mapping of locale strings (optional)

- If the built-in locale string mapping mechanism does not match your requirements, you can assign specific codepages to locale strings. See *Broker's Built-in Locale String Mapping* and `locale-string` for information on customizing the mapping of locale strings to codepages.

Configuring SAGTRPC User Exits

The user-written SAGTRPC user exit DLL must be accessible to the Broker worker threads.

To configure SAGTRPC user exits

1. Copy the user-written SAGTRPC user exit DLL into the EntireX *bin* folder.
2. In the Broker attribute file, set the service-specific or topic-specific broker attribute `CONVERSION` to the name of your SAGTRPC user exit. Example:

   ```
   CONVERSION=(MYRPC,TRACE=1)
   ```

   or

1. Place the user-written translation routine DLL in a folder of your choice. Spaces in the path name are not allowed.
2. In the Broker attribute file, set the service-specific or topic-specific broker attribute `CONVERSION` to the full path name of the folder of the SAGTRPC user exit. Example:

   ```
   CONVERSION="C:\MyDir\MyTrans\MyRpc.dll"
   ```

To configure locale string defaults

- If the broker's locale string defaults do not match your requirements, we recommend you assign suitable locale string defaults for your country and region. See the appropriate attribute under *Codepage-specific Attributes* under *Broker Attributes* for information on customizing broker's locale string defaults, and also *Locale String Mapping*.

To customize mapping of locale strings

- If the broker's built-in locale string mechanism does not match your requirements, you can assign specific codepages to locale strings. See *Broker's Built-in Locale String Mapping* and the appropriate attribute under *Codepage-specific Attributes* under *Broker Attributes* for information on customizing broker's locale string defaults.

Writing Translation User Exits

This section covers the following topics:

- Introduction
- Structure of the TRAP Control Block
- Using the TRAP Fields

Introduction

EntireX Broker provides an interface to enable user-written translation routines in the programming language C. It contains three parameters:

- The address of the TRAP control block (TRAP = Translation Routine / Area for Parameters).
- The address of a temporary work area. It is aligned to fullword / long integer boundary (divisible by 4). The work area can only be used for temporary needs and is cleared after return.
- A fullword (long integer) that contains the length of the work area.

Note:
Names for user-written translation routines starting with "SAG" are reserved for Software AG usage and must not be used, e.g. "SAGTCHA" and "SAGTRPC".

Structure of the TRAP Control Block

The C structure TR_TRAP covers the layout of the control block.

```
typedef struct _TR_TRAP                                    /* I / O */
{
  unsigned long  tr_type;             /* TRAP type: TRAP_TYPE  inp  */
#define TR_TYPE  2                    /* TRAP type ETB 121          */
  long           tr_ilen;             /* Input buffer length   inp  */
  unsigned char *tr_ibuf;             /* Ptr to input buffer   inp  */
  long           tr_olen;             /* Output buffer length  inp  */
  unsigned char *tr_obuf;             /* Ptr to output buffer  inp  */
  long           tr_dlen;             /* Len of data returned: out  */
                                      /* Minimum of tr_ilen         */
                                      /*        and tr_olen         */
  unsigned long  tr_shost;            /* Senders host          inp  */
#define TR_LITTLE_ENDIAN  0           /* little endian              */
#define TR_BIG_ENDIAN     1           /* big endian                 */
  unsigned long  tr_scode;            /* Senders character set inp  */
#define SEBCIBM ((1L << 5)|(1L << 1))  /* 0x22 EBCDIC (IBM)         */
#define SEBCSNI ((1L << 6)|(1L << 1))  /* 0x42 EBCDIC (SNI)         */
#define SA88591            (1L << 7)   /* 0x80 ASCII                */
  unsigned long  tr_rhost; /* Receivers host     (see tr_shost) inp  */
  unsigned long  tr_rcode; /* Receivers char set (see tr_scode) inp  */
  unsigned long  tr_bhost; /* BROKER host        (see tr_shost) inp  */
  unsigned long  tr_bcode; /* BROKER char set    (see tr_scode) inp  */
  unsigned long  tr_senva; /* Senders ENVIRONMENT field set:    inp  */
#define OFF  0                        /*  ENVIRONMENT field not set */
#define ON   1                        /*  ENVIRONMENT field set     */
  unsigned long  tr_renva; /* Receivers ENVIRONMENT field set:  inp  */
                           /* see tr_senva                          */
```

```
#define S_ENV 32           /* size of ENVIRONMENT field              */
  char           tr_senv[S_ENV];/* Senders   ENVIRONMENT field  inp  */
  char           tr_renv[S_ENV];/* Receivers ENVIRONMENT field  inp  */
} TR_TRAP;
```

The file *usrtcha.c* is an example of the translation user exit. It is delivered in the Broker user exit directory. See *Directories as Used in EntireX*.

Using the TRAP Fields

The `tr_dlen` must be supplied by the user-written translation routine. It tells the Broker the length of the message of the translation. In our example its value is set to the minimum length of the input and output buffer.

All other TRAP fields are supplied by the Broker and must not be modified by the user-written translation routine.

The incoming message is located in a buffer pointed to by `tr_ibuf`. The length (not to be exceeded) is supplied in `tr_ilen`. The character set information from the send buffer can be taken from `tr_scode`.

The outgoing message must be written to the buffer pointed to by `tr_obuf`. The length of the output buffer is given in the field `tr_olen`. The character set is specified in `tr_rcode`. If the addresses given in `tr_ibuf` and `tr_obuf` point to the same location, it is not necessary to copy the data from the input buffer to the output buffer.

The environment fields `tr_senva` and `tr_renva` are provided to handle site-dependent character set information. For the `SEND` and/or `RECEIVE` functions, you can specify data in the `ENVIRONMENT` field of the Broker ACI control block. This data is translated into the codepage of the platform where EntireX Broker is running (see field `tr_bcode`) and is available to the `tr_senv` or `tr_renv` field in the TRAP control block. `tr_senva` or `tr_renva` are set to `ON` if environmental data is available.

The sample source `USRTCHA` contains a section to handle the `ENVIRONMENT` value `*NONE`. The translation will be skipped if `*NONE` is supplied by the sender or receiver. Any values given in the API field `ENVIRONMENT` must correspond to the values handled in the translation routine.

Writing SAGTRPC User Exits

This section covers the following topics:

- Introduction
- Structure of the User Exit Control Block
- Using the User Exit Interface Fields
- Character Set and Codepage

Introduction

EntireX Broker provides an interface to SAGTRPC user exit routines written in the programming language C. The interface contains three parameters:

- The address of the UE (user exit) control block.
- The address of a temporary work area. It is aligned to a fullword / long-integer boundary (divisible by 4). The work area can only be used temporarily and is cleared after return.
- A fullword (long integer) that contains the length of the work area.

Note:
Names for conversion routines starting with "SAG" are reserved for Software AG usage and must not be used, e.g. "SAGTCHA" and "SAGTRPC".

Structure of the User Exit Control Block

The C structure UECB shows the layout of the user exit control block.

```
typedef struct _UECB
{
   unsigned long          eVersion;
#define USRTRPC_VERSION_1               1

   char                *  pInputBuffer;
   unsigned long          uInputLen;
   char                *  pOutputBuffer;
   unsigned long          uOutputLen;
   unsigned long          uReturnedLen;

   unsigned long          shost;
#define USRTRPC_LITTLE_ENDIAN  0      /* little endian                    */
#define USRTRPC_BIG_ENDIAN     1      /* big endian                       */

   unsigned long          scode;
#define USRTRPC_SEBCIBM ((1L << 5)|(1L << 1)) /* 0x22 EBCDIC (IBM)         */
#define USRTRPC_SEBCSNI ((1L << 6)|(1L << 1)) /* 0x42 EBCDIC (SNI)         */
#define USRTRPC_SA88591            (1L << 7)  /* 0x80 ASCII                */

   unsigned long          rhost;
/* see shost */
   unsigned long          rcode;
/* see scode */
   unsigned long          bhost;
```

```
/* see shost */
   unsigned long          bcode;
/* see scode */

   unsigned long          uCpSender;
   unsigned long          uCpReceiver;
   unsigned long          uCpBroker;

   char                   eFunction;
#define USRTRPC_FCT_CONVERT             'C'
#define USRTRPC_FCT_GETLENGTH           'L'

   char                   eDirection;
#define USRTRPC_DIR_SENDER_TO_BROKER    '1'
#define USRTRPC_DIR_SENDER_TO_RECEIVER  '2'
#define USRTRPC_DIR_BROKER_TO_RECEIVER  '3'

   char                   sFormat[2];
#define ERX_USERDATA    "01"    /* UserId, Lib, Pgm, etc. from Header
                                   (truncatable)                        */
#define ERX_METADATA    "02"    /* Header Data (non-truncatable)        */
#define ERX_FRMTDATA    "03"    /* Format Buffer (non-truncatable)      */
#define ERX_SB_ELEMENT  "04"    /* String Buffer                        */
#define ERX_VB_METADATA "05"    /* Value Buffer Array Occurences,
                                   String Length                        */
#define ERX_PREVIEW     "99"    /* Previewing FB and VB, etc...         */
                                /* Convert data lazy. Do not care on    */
                                /* length changes and truncation.       */
#define ERX_FRMT_A      "A "    /* Data Type A                          */
#define ERX_FRMT_AV     "AV"    /* Data Type AV                         */
#define ERX_FRMT_B      "B "    /* Data Type B                          */
#define ERX_FRMT_BV     "BV"    /* Data Type BV                         */
#define ERX_FRMT_D      "D "    /* Data Type D                          */
#define ERX_FRMT_F4     "F4"    /* Data Type F4                         */
#define ERX_FRMT_F8     "F8"    /* Data Type F8                         */
#define ERX_FRMT_I1     "I1"    /* Data Type I1                         */
#define ERX_FRMT_I2     "I2"    /* Data Type I2                         */
#define ERX_FRMT_I4     "I4"    /* Data Type I4                         */
#define ERX_FRMT_K      "K "    /* Data Type K                          */
#define ERX_FRMT_KV     "KV"    /* Data Type KV                         */
#define ERX_FRMT_L      "L "    /* Data Type L                          */
#define ERX_FRMT_N      "N "    /* Data Type N                          */
#define ERX_FRMT_P      "P "    /* Data Type P                          */
#define ERX_FRMT_T      "T "    /* Data Type T                          */
#define ERX_FRMT_U      "U "    /* Data Type U                          */
#define ERX_FRMT_UV     "UV"    /* Data Type UV                         */

   char                  szErrorText[40];

}  UECB;
```

The file *usrtrpc.c* is an example of the SAGTRPC User Exit. It is delivered in the Broker User Exit Directory. See *Directories as Used in EntireX*.

Using the User Exit Interface Fields

The user exit provides two separate functions, `Convert` and `GetLength`. The field `eFunction` indicates the function to execute.

Errors

Both functions can send an error, using register 15 in the range 1 to 9999 to SAGTRPC together with an error text in the field `szErrorText`.

- A value of 0 returned in register 15 means successful response.
- Error 9999 is reserved for output buffer overflow. See *`Convert` Function*.
- When an error occurs, the conversion of the message will be aborted and the error text will be sent to the receiver (client or server). The error is prefixed with the error class 1011. See *Message Class 1011 - User-definable SAGTRPC Conversion Exit*.

Example:

The user exit returns 1 in register 15 and the message "Invalid Function" in `szErrorText`. The receiver gets the error message `10110001 Invalid Function`.

Convert Function

This function has to be executed when the contents of `eFunction` match the definition `USRTRPC_FCT_CONVERT`.

`uReturnedLen` must be supplied by SAGTRPC's user-written conversion exit. Its value must be set to the length of the output buffer.

All other interface fields are supplied by the Broker and must not be modified by SAGTRPC's user-written conversion exit.

The incoming data is located in a buffer pointed to by `pInputBuffer`. `uInputLen` defines the length.

The outgoing converted message must be written to the buffer pointed to by `pOutputBuffer`. The field `tr_olen` defines the maximum length available.

For variable length data such as AV and KV, an output buffer overflow can occur if the message size increases after conversion or the receiver's receive buffer is too small. In this case error 9999 "output buffer overflow" must be returned, which calls the *`GetLength` Function* for the remaining fields.

GetLength Function

The `GetLength` function evaluates the needed length of the output buffer after conversion. An actual conversion must not be performed. The length needed must be returned in the field `uOutputLen`.

The `GetLength` function is called for remaining fields after the `Convert` function returned the error 9999 "output buffer overflow".

The purpose of this function is to evaluate the length needed by the receiver's receive buffer. This length is returned to the receiver in the ACI field `RETURN-LENGTH`. The receiver can then use the Broker ACI function `RECEIVE` with the option `LAST` together with a receive buffer large enough to reread the message.

Character Set and Codepage

The character-set information used is the same as in the user-written translation routine and is taken from `scode` (for the sender), `rcode` (for the receiver) and `bcode` (for the Broker). The character-set information depends on the direction information given in the field `eDirection`. See the following table:

eDirection	From Character Set	To Character Set
USRTRPC_DIR_SENDER_TO_BROKER	scode	bcode
USRTRPC_DIR_SENDER_TO_RECEIVER	scode	rcode
USRTRPC_DIR_BROKER_TO_RECEIVER	bcode	rcode

Alternatively, the codepage as derived from the locale string mapping process is provided in `uCpSender` (sender codepage), `uCpReceiver` (receiver codepage) and `uCpBroker` (Broker codepage), and can be used to find the correct conversion table. See the following table and also *Locale String Mapping*.

eDirection	From Codepage	To Codepage
USRTRPC_DIR_SENDER_TO_BROKER	uCpSender	uCpBroker
USRTRPC_DIR_SENDER_TO_RECEIVER	uCpSender	uCpReceiver
USRTRPC_DIR_BROKER_TO_RECEIVER	uCpBroker	uCpReceiver

Software AG IDL Data Types to Convert

The field `sFormat` provides the SAGTRPC user-written conversion exit with the information on the IDL data types to convert. Each data type can be handled independently.

sFormat	Data to be converted	Notes
FMTA	IDL data type A	1, 3, 4
FMTAV	IDL data type AV	4, 5
FMTB	IDL data type B	1, 2, 7
FMTBV	IDL data type BV	1, 2, 7
FMTD	IDL data type D	1, 2, 7
FMTF4	IDL data type F4	1, 2, 7
FMTF8	IDL data type F8	1, 2, 7
FMTI1	IDL data type I1	1, 2, 7

sFormat	Data to be converted	Notes
FMTI2	IDL data type I2	1, 2, 7
FMTI4	IDL data type I4	1, 2, 7
FMTK	IDL data type K	1, 3, 4
FMTKV	IDL data type KV	4, 5
FMTL	IDL data type L	1, 2, 7
FMTN	IDL data type N	1, 2, 7
FMTP	IDL data type P	1, 2, 7
FMTT	IDL data type T	1, 2, 8
FMTU	IDL data type U	1, 2, 7
FMTUV	IDL data type UV	1, 2, 7
FMTUSER	RPC user data such as user ID, library, program...	1, 3, 4
FMTMETA	RPC metadata	1, 2, 7
FMTFB	RPC format buffer	1, 2, 7
FMTSB	RPC metadata variable length	4, 5, 7
FMTPRE	Preview data	4, 6, 7

Notes:

1. Field length is constant.
2. The field content length must not increase or decrease during conversion. If this happens, the user exit should produce an error.
3. If the field content length *decreases* during the conversion, suitable padding characters (normally blanks) have to be used.
   If the field content length *increases* during conversion and exceeds the field length, the contents must be truncated or, alternatively, the conversion can be aborted and an error produced.
4. If the contents are truncated, character boundaries are the responsibility of the user exit. Complete valid characters after conversion have to be guaranteed. This may be a complex task for codepages described under *Conversion with Multibyte, Double-byte and other Complex Codepages*. For

Conversion with Single-byte Codepages it is simple because the character boundaries are the same as the byte boundaries.

5. The field length can decrease or increase during the conversion up to the output buffer length. The new field length must be returned in `uReturnedLen`. If the output buffer in the `Convert` function is too small, error 9999 must be returned to the caller.
6. The field buffer should continue to be converted until the output buffer is full or the input buffer has been processed. If the field content length increases or truncations occur, no error should be produced. If the field content length decreases, there should be no padding. The new field length should simply be returned to the caller.
7. Codepages used for RPC data streams must meet several requirements. See *Codepage Requirements for RPC Data Stream Conversions*. If these are not met, the codepage cannot be used to convert RPC data streams.

To compile and link the SAGTRPC User Exit

- See the *README.TXT* in the *Broker User Exit Directory*.

Building and Installing ICU Custom Converters

User-written ICU custom-converters can be used for *ACI-based Programming*, *RPC-based Components*, and *Reliable RPC*.

This section covers the following topics:

- Writing a User-written ICU Converter
- Compiling a User-written ICU Converter
- Installing a User-written ICU Converter

Writing a User-written ICU Converter

ICU uses algorithmic conversion, non-algorithmic conversion and combinations of both. See *ICU Conversion*. Non-algorithmic converters defined by the UCM format are the easiest way to define user-written ICU converters. See *UCM Format*.

To write a (non-algorithmic) user-written ICU converter

- Define the ICU converter file in UCM format using a text editor to meet your requirements.

  Note:
  For further explanation of the UCM file format, see *ICU Resources*.

Writing algorithmic and partially algorithmic converters can be complex. However, they can be installed into EntireX in the same way as the table-driven, non-algorithmic ones. A description of how to write algorithmic and partially algorithmic converters is beyond the scope of this documentation; please see the ICU documentation and other sources specified under *ICU Resources*.

Compiling a User-written ICU Converter

To compile the user-written ICU converter

- Compile the converter source files (extension *.ucm*) into binary converter files (extension ".cnv") using the ICU tool `makeconv`. Example:

  ```
  makeconv -v myebcdic.ucm
  ```

  Note:
  EntireX delivers the ICU tool `makeconv` in the EntireX *bin* folder.

  This produces a binary converter file named *myebcdic.cnv*.

  Caution:
  The binary format "cnv" depends on the endianness (big/little endian) and character set family (ASCII/EBCDIC) of the computer where it is produced. Under Windows, little endian ASCII binary converter files are produced. If the broker is running under Windows, it is highly recommended to compile the converter source file(s) under Windows also, otherwise unpredictable result may occur.

Installing a User-written ICU Converter

To install the user-written ICU converter

1. Check if the folder *Software AG* exists in the Windows local application data directory for all users. See *Application Data Directory*. If not, create the folder *Software AG*. Example:

   ```
   C:\Documents and Settings\All Users\Application Data\Software AG
   ```

2. Check if the folder *icudt<icu-version>l* exists in the folder *Software AG* (see 1 above) in the Windows application data folder for all users. If not, create the folder *icudt<icu-version>l* where:

   - *<icu-version>*is the ICU version used, for example "32".
   - "l" stands for little-endian, example:

   ```
   C:\Documents and Settings\All Users\Application Data\Software AG/icudt32l
   ```

   Notes:

   1. The folder and its naming are given by ICU standard. It is not invented by Software AG.
   2. See the Release Notes to determine the ICU version used by the broker you are running and form the correct folder name - otherwise the user-written ICU converter will not be located.
   3. There are also other approaches supported by ICU to locate converters. These approaches are (also) ICU version dependent. However, Software AG recommends the mechanism described above. See the ICU website for more information under *ICU Resources*.

3. If the converter name is not sent as the locale string by your application, customize the mapping of locale strings by assigning the user-written ICU converter (codepage) to locale strings in the Broker attribute file, see `locale-string` for how to customize the mapping of locale strings to codepages. Example:

```
DEFAULTS=CODEPAGE
/* Customer-written ICU converter */
CP1140=myebcdic
CP0819=myascii
```

4. For the Broker attribute, check whether ICU conversion is possible, that is, the attribute `ICU-CONVERSION` is
   - not defined, its default is "YES"
   - set to "YES"
5. For the Broker attribute, check whether use of ICU custom converters is possible, that is, the attribute `ICU-SET-DATA-DIRECTORY` is either
   - not defined, its default is "YES"
   - (is) set to "YES"