



The language constants are defined in e32const.h.

Define the languages that your component is localized to in your .mmp file:

```
// file: MyApp.mmp
...
LANG 01 02 16 // UK English, French, Russian
```

Place logical strings for different languages in different language-specific files. Dont forget to define the file encoding for those languages that use character sets beyond basic ASCII. For instance:

```
// file: MyApp.l01 (ELangEnglish = 01)
#define message_wait      "Please wait..."
#define mess_with_param   "You have %N new messages."

// file: MyApp.l02 (ELangFrench = 02)
// the file has to be encoded in UTF-8 for the strings to be rendered correctly:
CHARACTER_SET UTF8
#define message_wait      "Attendez, s'il-vous plait..."
#define mess_with_param   "Vous avez %N nouveaux messages."

// file: MyApp.l16 (ELangRussian = 16)
// the file has to be encoded in UTF-8 for the strings to be rendered correctly:
CHARACTER_SET UTF8
#define message_wait      "???????????, ?????????? ..."
#define mess_with_param   "????????????? ?????? ???????????: %N"
```

In file MyApp.loc, include the correct language-specific file:

```
// file: MyApp.loc
CHARACTER_SET UTF8

#if defined LANGUAGE_01
    #include "MyApp.l01"
#elif defined LANGUAGE_02
    #include "MyApp.l02"
#elif defined LANGUAGE_16
    #include "MyApp.l16"
#endif
```

In MyApp.rss, define the resources based on the strings defined in the language-specific file that was included by MyApp.loc:

```
// file: MyApp.rss
#include <eikon.rh>
#include "MyApp.loc"

RESOURCE TBUF r_message_wait { buf = message_wait; }
RESOURCE TBUF r_mess_with_param { buf = mess_with_param; }
```

How_to_define_localization_messages

In your code, use the localized strings as follows. The string denoted by R_MESSAGE_WAIT will be displayed in the appropriate language. Note the change from lower-case to upper-case.

```
//file: testexample.cpp
#include <stringloader.h>
...
...

// how to load simple message
HBufC* message = StringLoader::LoadL( R_MESSAGE_WAIT );

// how to load message with params
TInt msgCount = 10;
message = StringLoader::LoadL( R_MESS_WITH_PARAM, msgCount ); // You have 10 new messages.

CAknInformationNote* note = new ( ELeave ) CAknInformationNote( ETrue );
note->ExecuteLD( *message );
delete message;
```

Finally, add the localized language resources to the package file:

- if you want to support only current language of the device, use following approach:

```
;; file: MyApp.pkg
...

; languages supported:
&EN,FR,RU

; localization:
{
"${(EPOCROOT)epoc32\data\z\resource\apps\MyApp.r01"
"${(EPOCROOT)epoc32\data\z\resource\apps\MyApp.r02"
"${(EPOCROOT)epoc32\data\z\resource\apps\MyApp.r16"
}-"!\resource\apps\MyApp.rsc"
```

- if you want to allow user select necessary languages, you could use following approach:

```
;; file: MyApp.pkg
...

; languages supported:
&EN,FR,RU

; localization:

!({"English", "English", "English" }, { "French", "French", "French" }, { "Russian", "Russian", "R

IF (Option1)
    "${(EPOCROOT)epoc32\data\z\resource\apps\MyApp.r01}-"!\resource\apps\MyApp.r01"
ENDIF

IF (Option2)
    "${(EPOCROOT)epoc32\data\z\resource\apps\MyApp.r02}-"!\resource\apps\MyApp.r02"
ENDIF

IF (Option3)
```

How_to_define_localization_messages

```
"$(EPOCROOT)epoc32\data\z\resource\apps\MyApp.r03"-"!:\resource\apps\MyApp.r03"  
ENDIF
```

There are two ways to add localized resource files into a .pkg file and generate the SIS package. Read [Knowledge Base article](#) for details.