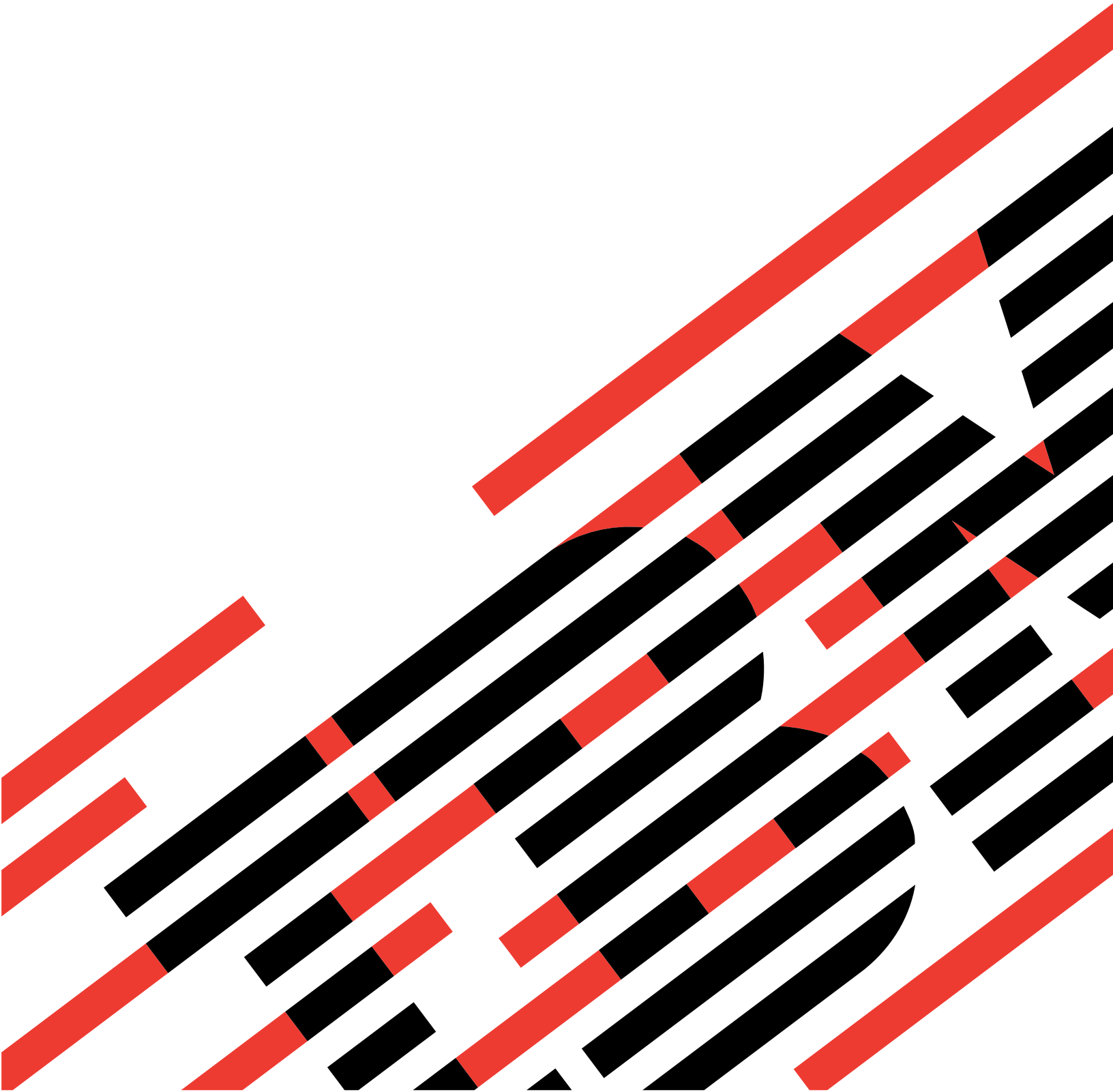
IBM

eserver

iSeries

Globalization (Develop global applications)



IBM

@server

iSeries

Globalization (Develop global applications)

Contents

OS/400 globalization

As companies integrate e-commerce on a global scale into their fundamental business processes, their prospective customers, established customers, and active partners can take advantage of increased revenue and decreased expenses through software globalization. They also can improve customer communications and increase savings. Globalized software gives you the following advantages:

- Increased customer satisfaction that can increase sales
- Enhanced customer support communications
- Enhanced global information dissemination
- A better return on Information Technology (IT) investments

This information shows you how to:

- Create an application efficiently and at minimal expense.
- Retrofit existing applications for globalization and create new applications designed for globalization. Designing an application for globalization, however, is usually less expensive than retrofitting an existing application.
- Ensure that the application design does not interfere with the current or planned design of other internationalized applications.

These pages provide a single source for the information you need to build applications for national and international audiences. You can also find information about what's new in this release and how you can print this topic.

Globalization overview
This topic describes the way that globalization has been implemented on OS/400(R), including topics that describe globalization-specific values on the system and topics that describe how services and functions in OS/400 support globalization.

Set up OS/400 with a national language version
This topic describes the steps you need to take to properly install and configure a national language version on OS/400, with topics ranging from selecting and installing hardware, installing software, and configuring your environment to run in a globalized setting. You can use this information as you install your own servers, and you can apply the principles when you develop applications for customers who are installing their own national language version on OS/400.

Develop global applications
This topic provides guidelines for designing, developing, and delivering globalized applications:

- Designing functions that are sensitive to national languages
- Supporting various types of hardware support
- Translating the textual data in your application
- Making your application available worldwide

Handle data in globalized applications
This topic describes the ways in which OS/400 enables you to handle data in a globalized environment. Included in these pages are topics that describe Unicode and UCS-2 data, the Chinese standard GB18030, how to use CCSIDs to integrate multiple language environments consistently, and how to use bidirectional data, DBCS data, and locales.

Globalization reference information
This topic provides detailed supporting information for the concepts and tasks discussed in the Globalization category.

Globalization checklists
This topic gathers together all of the checklists that are contained with these pages. These checklists are useful reminders of the issues you need to consider as you create and work with global applications.

Develop global applications

Global applications are those applications that have national language support. National language support allows users to enter, store, process, retrieve, print, and display data in their language of choice. National language support also allows users to see and enter data, commands, prompts, messages, and documentation in their language of choice, in formats that match their cultural expectations.

Although your reasons may differ, most internationalized applications are created because:

- The market demands global software products that have a local feel
- The application is used in a community that represents multiple cultures
- Revenue opportunities are expanded

The following links provide valuable information that you need to know as you begin your development process:

- Develop global applications: goals and processes
- Design globalized applications
- Programming considerations in global application design
- Deliver globalized applications

For more information

See Handle data in globalized applications for information about how you work with various types of data in a global environment.

Goals and processes

Before you invest your time and money into the development of global applications, you will benefit from a planning process that gives you an opportunity to consider how you can efficiently and effectively serve your global users. The following topics will help you develop such a plan:

- Development goals
- Market research process
- Development process
- Documentation process
- Translation process
- Testing process
- Packaging and installation process
- Application maintenance process

Globalization development goals

Use this topic when you are planning for, and creating, international applications. The recommendations in this topic assume that your basic goals are:

- To create an application efficiently.
- To create an application at minimal expense. You can retrofit existing applications for globalization and create new applications designed for globalization. Designing an application for globalization, however, is usually less expensive than retrofitting an existing application.
- To ensure that the application design does not interfere with the current or planned design of other internationalized applications.

- When creating an application with national language support, you must plan for or put into effect the following tasks:
  - Designing functions that are sensitive to national languages
  - Supporting various types of hardware support
  - Translating the textual data in your application
  - Making your application available worldwide.

Globalization development planning processes

A global application should be well planned at every stage in order to save time, effort, and money. You should not have to recompile programs nor repackage data objects. Your product may, however, be required to use a different data object based on the language version you are using. You should have one set of program code and different sets of culture- and text-dependent code, as needed.

Consider the following processes when planning for a global application.

- Market research
- Development
- Documentation
- Translation
- Testing
- Packaging and installation
- Application maintenance

Market research process

The most important factor for every decision is that you know for whom you are designing and developing your applications. To determine the answer to this question, ask yourself and your potential customers the following types of questions.

What are my target markets for today and tomorrow?

The answer to this question makes a significant difference if you define your marketplace in different countries or only in the area of your own language, or if you decide to include countries speaking other languages. For example, if you are coding an application from a Latin-based language, application complexity increases when you decide to include countries using non-Latin languages such as Hebrew, Chinese, or Japanese. The application complexity increases because you need to deal with incompatible characters sets and more complex input methods.

Along with the language problem, there are other areas to consider. You need to understand the culture, habits, ways of doing business, and laws of the target markets. You need to understand the customers' ways of life for you to be accepted as a business partner, to be able to get into the market, and to support them in their countries.

These factors can affect:

- The skills that you need (technical, cultural, language, laws)
- The environments to consider
- Your company structure and support organization
- Your relationship to other companies
- The resources that you need (people, time, and money)

Who are the users of my application?

You must understand the requirements that future users of your application will have. For example, do they want to:

- Work with separate databases for different languages?
- Work with a shared database for all languages?
- Exchange or consolidate data?
- Work with different languages dependent on the end user, the company, or the company's customers?
- Use end-user database tools to do their own inquiries on the application database?

All these factors may affect the design you choose, the way your application is able to switch from one language environment to another, and how data presentation and conversion take place.

How much globalization support is needed?

After you understand the requirements for your customers and their end users, you can decide what kind of culture-sensitive information you need to store and maintain, the type of data presentation, which parts you have to translate, and how your application must be able to be integrated in the different environments.

What is the cost of the effort?

To estimate the expected revenue, analyze the places you have chosen as your target market. After you know the requirements, you should be able to determine the effort and costs. This amount allows you to compare the costs against the expected revenue.

Which costs more, enabling or retrofitting an application?

The initial cost of enabling an application for national language support might be higher. But consider that the enabling steps are based more on normal modular and data-driven design techniques, which improve the quality of your application even without NLS enabling. Because a good design helps people to understand and describe the application system, you will receive a certain return on the investment. A good design helps to improve productivity of development and maintenance. You have the additional effort of designing and implementing the application only once, even for many different language versions. Compared to retrofitting an existing application, it is much less expensive to plan and design it from the very beginning.

Development process

Before you are ready to develop NLS-enabled applications, consider the following for a successful development process.

Education for developing internationalized applications

When you intend to develop NLS-enabled applications, you need to consider additional initial education. The following are important topics to learn about:

- General globalization concepts
- Available globalization support on OS/400
- Available globalization support on other systems and applications with which your application operates
- Isolation of different parts of an application
- Data presentation corresponding to cultural conventions
- Design and coding for textual data parts
- Translation process
- Product and system integration
- Packaging, installation, and setup

- Product support and maintenance

Based on the globalization enabling guidelines, first prepare a prototype application and test the chosen way of implementing the application for your specific environment. Afterward include the globalization enabling guidelines in your general application development processes, guidelines, and standards.

Implementing internationalized applications

When implementing an internationalized application, the most important objective is to produce only one set of running code. You must differentiate consistently between running code and textual data. It is essential that you standardize the chosen approach throughout the whole application. Work with unique and clearly defined naming conventions. To understand and to maintain this information in the application, handle parameters called from a program in a consistent way.

Documentation process

Documentation should provide information for the end users of the application system in their own language. The documentation should also include installation, setup, and customization information for the end user, the system operator, and the application system manager.

The user documentation should be textual data that can be easily translated. Whenever possible, combine the online help information and user documentation to reduce the volume of text to translate. Any example displays or print layouts should be produced by the application and included in the documentation.

Translation process

Translating the textual data is a very time-consuming process. The textual data should be available to translators very early in the development stage, even before the code is stable. Consider the following areas when planning for translation:

Physical equipment

Each translator should have equipment compatible with the language being translated. The display stations and keyboards should have all the characters needed to translate, and the printers should be able to print the translated text.

Translation tools

Provide the translators with tools that increase productivity and that prevent translation of non-textual application data. When purchasing or developing a translation tool, the following features should be included.

- An editor that provides the ability to show displays that would be seen by the end user, and the ability to translate the textual data on the system but still protect the parts of the application that are not textual data. The editor should also include functions such as scan and replace, find, copy, move, and delete.
- A dictionary function to provide consistency of words and phrases throughout the product.
- A validation process to check translation errors that might cause the application to malfunction.
- A merge function that provides the ability to merge the translated text into a new version of the original text. This merge function allows for translating only new text, and saves time and effort.
- A print function for validation purposes.

Translation education

It is important that translators are familiar with the product they are translating and also with the tools they are using. The translation process is not the replacement of one word with another, but the formation of concepts in another language. Knowledge of the product being translated provides more understandable products to the end user. Time and resources for educating translators should be planned well in advance.

Translation guidelines and instructions

Translation guidelines and instructions should be provided to ensure correct translation. For example, to translate an error message properly, it is important to know in what context this message is displayed. A note to translators telling them what error caused the message to be displayed also helps.

Translation glossary

To ensure accurate translation, use terminology based on definitions in standard, widely available, dictionaries. If your application uses terms not found in standard dictionaries or terms that are used differently from standard definitions, provide a glossary of non-standard terms to the translators. Avoid using abbreviations and acronyms in your application. If you must use abbreviations or acronyms in your application, define them in the glossary. Remember, abbreviations and acronyms that are obvious in your language may not be obvious in another language.

Testing process

The testing of an globalization-enabled product should be done in three phases:

1. Testing the running code
   The running code should be tested in a globalization support environment in order to check all the possible language-dependent combinations. Translators should not test the product functionality.
2. Checking the textual data
   The textual data should be tested to check correct translation and consistency throughout the product.
3. Integrating the running code and textual data
   After the textual data and the code have been tested separately, an integration test should be performed to test if the application has taken into account all the globalization-related processing, and that the translation of the textual data has not caused a malfunction in the product.

   If your application will also run on a multinational or multilingual system a separate test that includes more than one set of textual data should be planned.

Packaging and installation process

Consider running code, translated textual data, and installation documents when packaging applications. Some suggestions for simplifying the packaging and installation of your application include:

- Store the running code and textual data separately.
- Package the textual data so that customers receive only the textual data in the languages that are ordered. (If the textual data for all languages is sent to all customers, it will waste system resources and lead to maintenance problems.)
- Provide comprehensive installation documents (translated to the language of the person installing the product) to avoid unnecessary operator-related problems and also to avoid the wrong impression right at the beginning that the application is not reliable.

  Installation documentation should cover the following topics:
  - What is needed to install and run the application, such as hardware and software requirements.
  - How to install the application, and how to recover when things go wrong.
  - What changes need to be made regarding:
    - Subsystem definitions
    - Device descriptions
    - User profiles
    - System values
    - Library lists
  - What are the application limitations?

Application maintenance process

Consider the following points when planning for maintenance of a multilingual application:

- The running code must be maintained separately from the textual data. These separate components must be fully synchronized. A redesign in one component may cause a redesign to be made in another.
- Whenever textual data is changed, be sure that it is incorporated in all the languages to which your textual data was translated. In this way, you can ensure a single maintenance level for the complete product.
- Be sure to test the running code for each textual data change that you distribute.

Design global applications

Your goal in designing international application components is to create components that support national languages independently. The support of one language should not interfere with the support of another language. The support of one language should not force any reduction in the function of the product for another language.

Your application should be able to support multiple languages simultaneously. For example, support for a double-byte coded character set (DBCS) language should not exclude support for single-byte coded character set (SBCS) languages. When you set up your libraries, consider using multiple textual data libraries, which can be dynamically allocated for testing, packaging, and delivery.

As you develop a global application for the iSeries server, you must consider these and other unique design issues that will affect the way you build and code your application for the global user. The following topics identify the scope of these issues, and provide useful guidance on how you should proceed:

- Checklist: Application design
- Globalization and localization
- Application arrangement and architecture
- User interfaces

Checklist: Application design

The following table provides some guidelines that you can follow when creating an application with national language support.

Complies	Not applicable	Rule
		The existence of a specific character set within a system or its components must not be assumed.
		Converting character case must be definable for each language and code page.
		Folding must be definable for each language and code page. Folding is the process in which characters that can be printed or displayed are substituted for those that cannot be printed or displayed on a particular device.
		The use of a graphic character for software control purposes must not preclude the use of the same character in the text of messages, menus, prompts, input fields, or output fields.
		The set of characters allowed for use in the entry of data must be definable by the system operator, a user, or an application.
		Graphic symbols and icons must be translatable.
		All characters on the active code page must be accessible.
		Language-dependent parts of a product must be isolated from non-language-dependent parts for easy modification.
		The design of a product must allow for the national language support of the various components of the product to be independent of each other.

Complies	Not applicable	Rule
		National language exits must be provided at strategic points.
		Diagnostics must be enabled.
		Logical layouts different from a given physical keyboard layout must be available to the user.
		All user interface text and presentation control information must be isolated from the running code.
		Functions dependent on display field length and display field position, or display field position alone, must not be designed in such a way that they are affected by user-interface text expansion.
		A method must be provided to allow for the identification and tracking of panels and messages during the translation process.
		Variables must be permitted to assume any location and order within a display field.
		Messages and other displayed words or phrases must be complete entities and must not be constructed from individual words or phrases.
		Entry of end-user commands, keywords, or responses must be possible without regard to uppercase or lowercase characters.
		A product with national language-dependent functions must be designed to facilitate the addition of other countries or national languages.
		Lowercase alphabets should not be assumed to be invariant.
		Character sets should be definable by the operator, a user, or an application.
		Special characters, including punctuation marks, should be definable and not program dependent.
		User-interface text modules should be packaged separately from the running code.

Globalization and localization

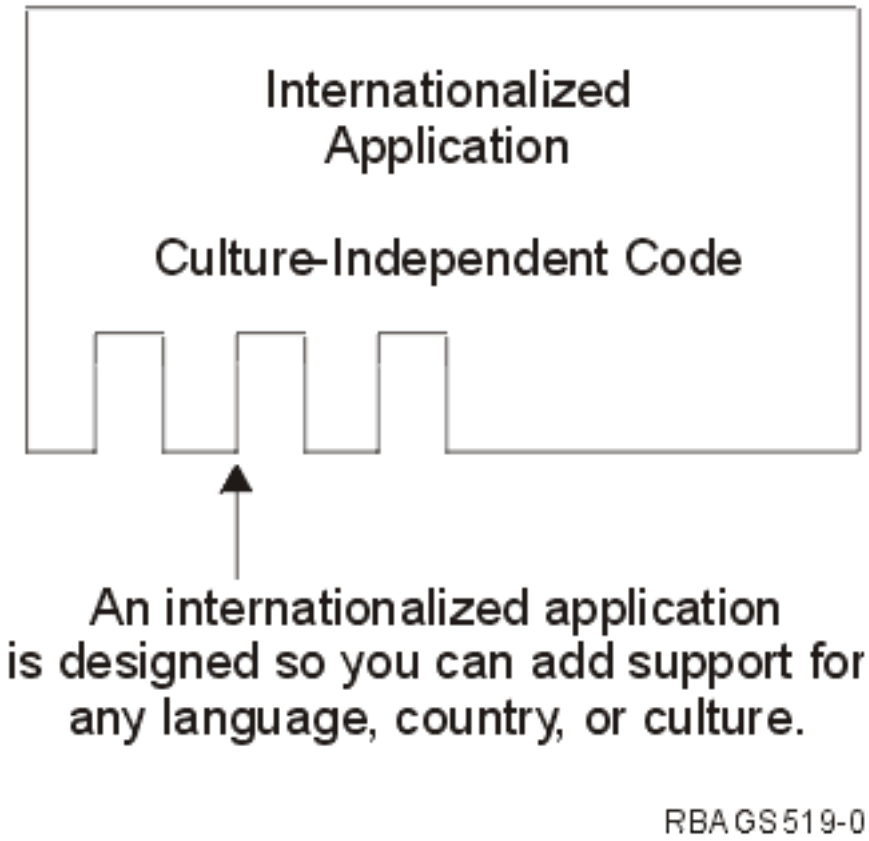
OS/400 controls the operation of programs and provides services such as controlling resources, scheduling jobs, controlling input and output, and managing data. It is designed to complement and extend the capabilities of iSeries servers to provide fully-integrated support for interactive and batch applications.

Many OS/400 functions apply directly to interactive data processing. Among these functions are:

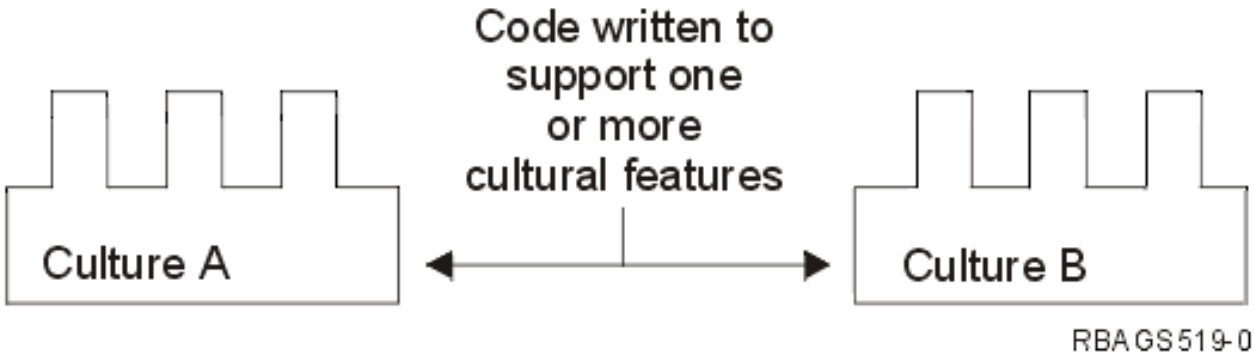
- Database support to make up-to-date business data available for rapid retrieval from any workstation
- Work management support to schedule the processing of requests from all work station users
- Application development support that allows online development and testing of new application programs to run at the same time as normal production activities
- System operation support that allows the user responsible for system operations to perform work from the display station using a single control language, complete with prompting and help for all commands
- Help and index search support that allows users to request online information on a wide variety of topics
- Message handling support that allows communication among the system, the user responsible for systems operations, workstation users, and programs running in the system
- Security support to protect data and other system resources from unauthorized access

In addition to these functions, the OS/400 program provides national language support. National language support allows users to interact with the system in the language of their choice, with results that are culturally acceptable. National language support consists of two parts: globalization and localization.

Globalization is support that allows an application to operate in all language environments without any change to the application. This type of design is also known as enabling an application for national language support. A globalized application, shown in the following figure, is culturally neutral.

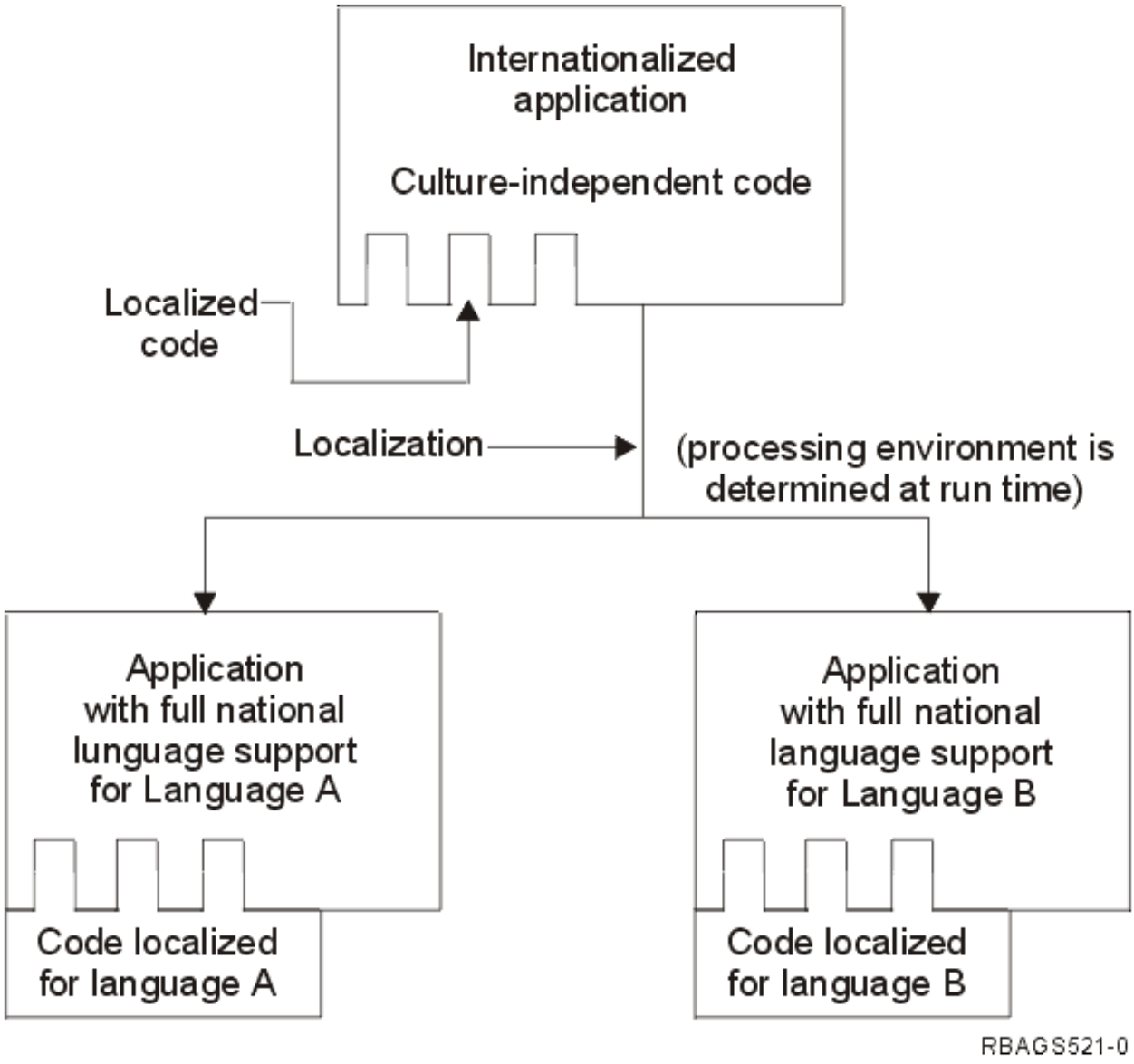


By contrast, **localization** allows an application to operate in a specific language, country, or culture. Localization of an application goes a step beyond globalization of the application, as shown in the following figure.



When localized code is integrated with globalized code at run time, the resulting application appears to the user with full national language support. The processing environment defines which localization code is

combined with the globalized code at run time, as shown in the following figure.

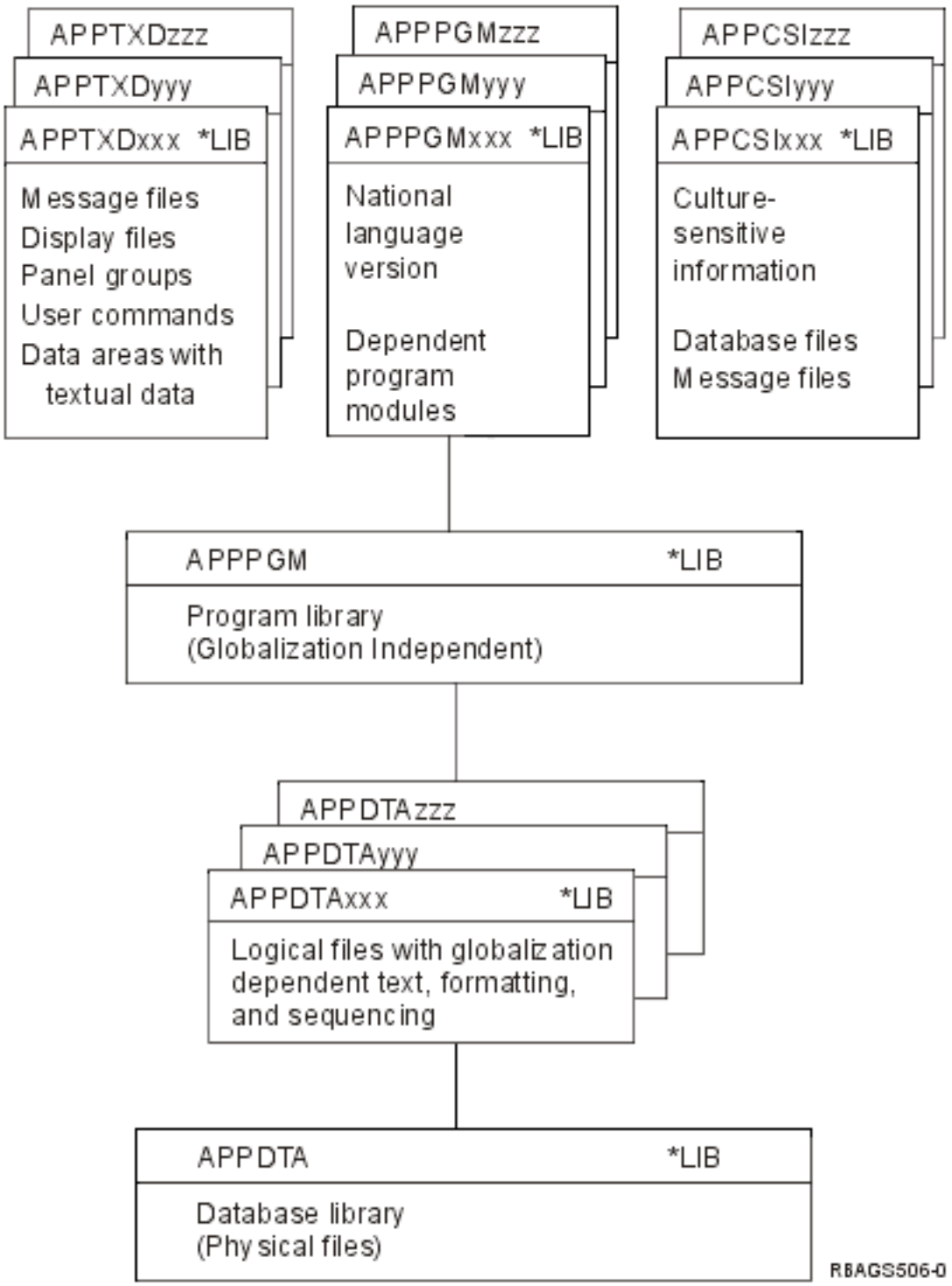


Application arrangement and architecture

When you design an international application, consider the ways that you can organize and structure your application so that it can be used in an international environment. In particular, consider the following strategies:

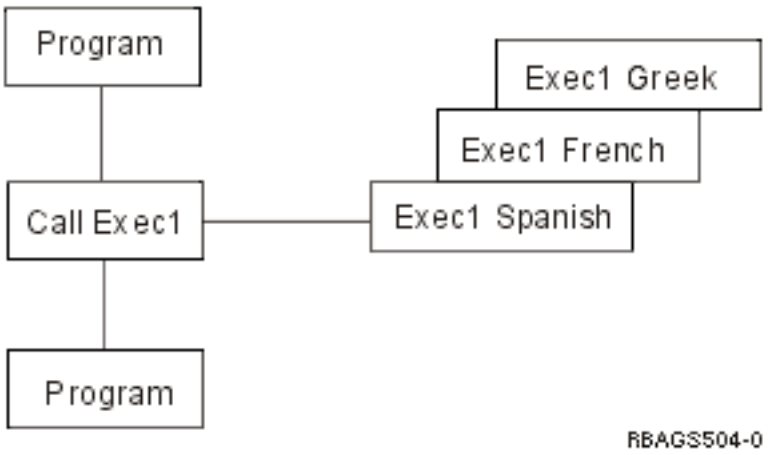
- Separate program modules at appropriate places
- Name application parts appropriately for a multilingual environment
- Refer to specifications whenever possible
- Provide multiple sets of logical files in separate libraries when working with database definitions

The following figure shows you the recommended way to organize the parts of your application.



Program module separation: You can separate culture-dependent parts from your running code and set up culture-dependent environments. You can do this using system values, user profile attributes, job attributes, and object attributes.

When it is impossible to separate national language and culture-dependent parts from the running code, you must provide national language exits or calls at all points where functions dependent on national language support are required. The following figure shows a national language exit.



Application part names: When you want to enable your application for different languages and countries, consider the environments of the target systems in your naming conventions. Use characters that are available, can be displayed, and can be printed in all the target environments. Use only characters of the invariant character set whenever you specify names for:

- Libraries
- Database files
- Device files (display or printer)
- Help panels
- Message files
- User commands
- Programs
- Record formats
- Fields

All other characters either vary their meaning or may not be available on the keyboard.

To create an internationalized application, you need to divide your application objects into related parts that are textual data and nontextual data. Your naming conventions should be able to distinguish between these parts. You should also be able to distinguish between the textual data of different languages. You can do this by separating the objects into different libraries.

Scenario: Library naming convention

Your library naming convention could look like the following:

```
AAATTTLLL
```

where: **AAA** is the application identification; **TTT** is the type of objects; and **LLL** is the language code.

This naming convention allows you to have all libraries that belong to an application grouped together because you have a unique identifier (AAA) at the beginning.

The second part (TTT) allows you to distinguish between different types of objects:

Textual data

- Display files
- Printer files

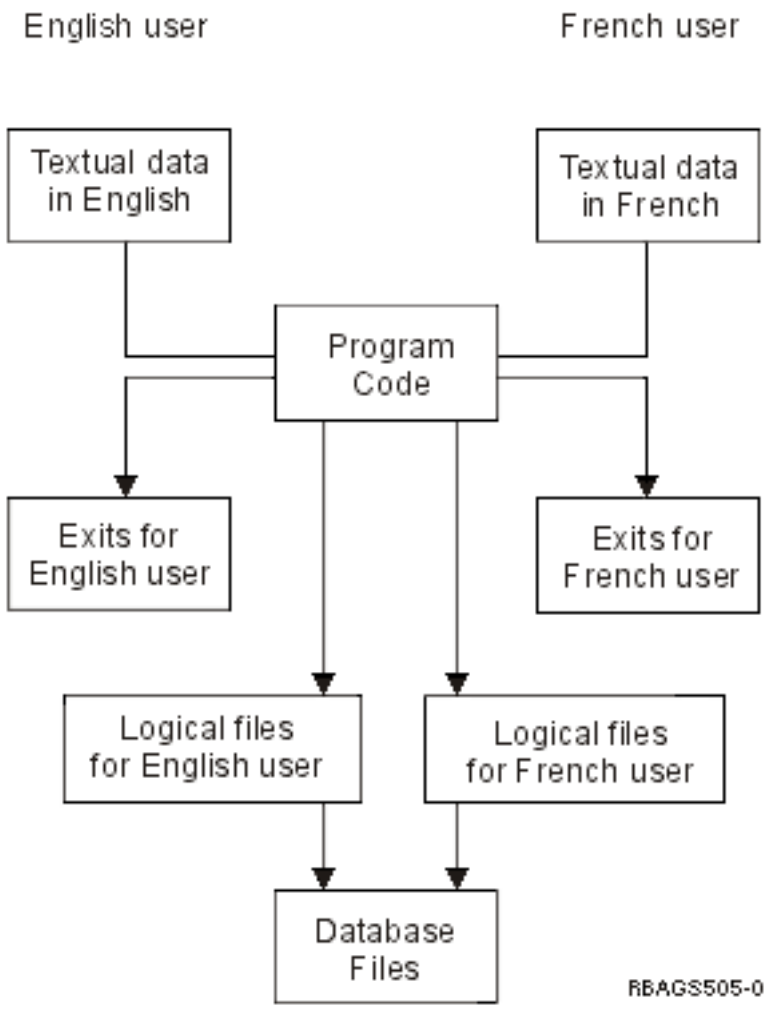
- Message files
- Help panels
- User command
- Cultural values
- Database files with NLS-sensitive information and specifications
- NLS-dependent program modules

Nontextual data
Programs

Data Database files

The third part (LLL) allows you to specify the national language version for all the textual data parts. This allows you to use the same names for objects of the different national language versions within the different libraries. Your program is able to use different objects by just rearranging the library list accordingly when the job is run.

The initial library list can be taken from the job description. You can build a new library list by specifying the library list in the INLLIBL parameter of the Create Job Description (CRTJOBD) command for a new job description, or the Change Job Description (CHGJOBD) command for an existing job description. The following figure shows an example of this.



Specification references: Define all your fields first in the field reference file of your application and refer to them whenever you can: in the database specifications, in device file specifications, and in the high-level language programs. This technique helps you to define the field specifications once and use them again. If you need to distinguish between the same field of different sources, you can rename or qualify them. Whenever you need to change the definition of a specific field, you just need to change the attributes of that field in the field reference file and create the objects again. Then the changes take place automatically in all the different places where the field is used.

For example:

```
|...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8
     A                                     REF(field-ref-file-name)
     A          R record
     A field    R line  pos
or
     A field    R line  pos                REFFLD(ref-field-name)
```

Database definitions: You define a file to specify certain facts, and the specifications are then used on database files. The following are some examples of such specifications:

- The object description text of the file
- The explanation text (TEXT keyword) on record formats and field descriptions
- The column headings (COLHDG keyword) on field descriptions
- Date and time formats and separators
- Sort sequence
- Language identifier

The object description text is shown by all database tools such as DB2(R) UDB for iSeries SQL, iSeries Access, and data file utility (DFU) on the file selection display.

The column headings are shown by the database tools on the output field definition display. Column headings are also used on screen design aid (SDA) and report layout utility (RLU) as the proposed field-prompting text or heading.

Data management handles date- and time-type fields in the format specified at file-creation time, unless your application or database tool does a conversion to present it according to your request or job demand.

When you want to present all this information according to the language and culture of the user, you need to provide multiple sets of logical files in separate libraries. Along with the translated text, you can specify different date and time formats or different sort sequence and let data management perform the conversion. A similar technique can also be used for numeric-type date fields (unless they are packed), using the substring (SST) function. The user can access the data only through the designated logical views. When you are defining logical files with different sort sequences, avoid using a unique index with a shared-weight table. Although this is possible, a unique index prevents using keys that differ only in characters with the same weight.

The scenario in Application part names shows an example of using different sets of logical files for different users.

User interfaces

A user interface is the part of a software product that your customer actually sees. A user interface may include the layout of display screens or printed output, displayed or printed text, commands, online help, and messages. A user interface is also the part of a software product that you must either translate or make cultural changes to for users in other countries or cultures.

OS/400 provides specific software functions to help you organize text from your user interface and store that text in a library for easy translation. The operating system also provides you with a user interface

manager that provides a consistent user interface. The user interface manager provides comprehensive support for defining and running panels such as displays and online help.

This section provides guidelines that you can follow when designing a user interface for an international application. You should apply these guidelines early in the design process. Guidelines are provided for:

- Checklist: User interface design
- Text translation design
- Textual data code design
- User interface manager
- Program message design
- Menu design
- Command design
- Cultural-dependent design
- Display file design
- Printer file design and translation
- Source file design
- CDRA design
- Handling languages that do not have NLV support

Checklist: User interface design: When creating a user interface with global support, you should follow some rules and guidelines, as shown in the following table:

Complies	Not applicable	Rule
		The use of a graphic character for software control purposes must not preclude the use of the same character in the text of messages, menus, prompts, input fields, or output fields.
		Graphic symbols and icons must be translatable.
		Language-dependent parts of a product must be isolated from nonlanguage-dependent parts for easy modification.
		All user interface text and presentation control information must be isolated from the running code.
		Sufficient space must be available for user-interface text expansion caused by translation.
		Functions dependent on display field length and display field position, or display field position alone, must not be designed in such a way that they are affected by user-interface text expansion.
		A method must be provided to allow for the identification and tracking of panels and messages during the translation process.
		Variables must be permitted to assume any location and order within a display field.
		Messages and other displayed words or phrases must be complete entities and must not be constructed from individual words or phrases.
		Entry of end-user commands, keywords, or responses must be possible without regard to uppercase or lowercase characters.
		Date and time formats must be selectable.
		Numeric punctuation must be selectable.
		Number rounding and mathematical formats must be selectable.
		Monetary format must be definable.

Complies	Not applicable	Rule
		The default currency symbol and its abbreviations must be selectable.
		The currency symbol position must be selectable.
		Field sizes for monetary values must be selectable.
		The measurement system must be selectable.
		Lowercase alphabets should not be assumed to be invariant.
		Special characters, including punctuation marks, should be definable and not program dependent.
		User-interface text modules should be packaged separately from the running code.
		User-interface text modules for single-byte coded character set systems should be loaded separately from the running code.
		A consistent convention should be used throughout the product for denoting variables and input fields.
		Words should not be used in place of numbers.
		The terminology in user interface text should be consistent throughout a product.
		Abbreviations should be avoided.
		Slang, jargon, and humor should not be used.
		Trademarks should be identified and explained.
		Ambiguous words should not be used.
		Proper style and sentence structure should be used in user interface text.
		Negative questions should be avoided.

Text translation design: The following information provides some general tips to help simplify the translation of your textual material.

Isolate textual data from running code

To allow easier translation and to avoid translating the running code, you should separate all textual data from the running code. Only one set of running code is needed, but many translations of the textual data can be done.

Provide expansion space

The space needed to translate text from one language to another varies by language. To ensure that the translated version preserves the concept and keeps usability, allow sufficient presentation space for the textual data expansion. The following table shows recommended expansion space for user interfaces designed using U.S. English.

Number of characters in text	Additional space required
Up to 10	100 to 200%
11 to 20	80 to 100%
21 to 30	60 to 80%
31 to 50	40 to 60%
51 to 70	31 to 40%

Number of characters in text	Additional space required
Over 70	30%

Variable placement of an object on the display

Because the position of one display element often is influenced by the position and size of others, some of the elements on the translated version of a display may have to be relocated. The program must continue to respond properly, despite this relocation.

Flexible order of variables

In order to contain dynamic information, messages usually employ substitution variables. However, each spoken language has its own syntax (order of arrangement of parts of speech). When a message is translated into another language, the position and order of substitution variables may have to change to meet the syntax requirements in the translated language.

Complete textual data entities

If the final form of the constant text relies on the composition of various parts, it may be untranslatable. This is because the translator might not know which form of the word to use or because there is no combination of parts that work for a different language.

For example, you should define column headings for display screens as complete entities. You should not combine words or parts of words to define column headings. Assume you are writing an application for scheduling jobs between Monday and Friday. You are creating your application in French. You decide to create column headings for reports and screen displays by combining the first part of the name of the day with the constant DI. Throughout the application, the column and report headings are assembled like this:

```
First Part of the
Name of the Day:    Combine With:      Result:

LUN                  DI            LUNDI
MAR                  DI            MARDI
MERCRE               DI            MERCREDI
JEU                  DI            JEUDI
VENDRE               DI            VENDREDI
```

When you translate your application from French to German, you cannot combine two parts to create the names of the days: MONTAG, DIENSTAG, MITTWOCH, DONNERSTAG, and FREITAG.

Treat commands, responses, and keywords like textual data

Commands, responses, and keywords should be translated into the language normally spoken by the user. For example, an English application has been translated into German. If the response is still in English as Yes and No, the German users would feel unfamiliar and uncomfortable in using the program because the responses they are familiar with are Ja and Nein.

Express all text as simply and clearly as possible

- Use simple phrases and sentences and avoid compound phrases. Simple words allow easy translation.
- Make terminology consistent throughout the product.

  If consistent terminology is not being adopted throughout the product, translators will waste time trying to determine the appropriate word to be used in translation.
- Include notes to translators in your information for correct word use to prevent any misunderstandings.
- Avoid abbreviations.

Rules for abbreviations vary from language to language. Abbreviations of words can lead to misunderstandings by the translator and by the end user.

- Avoid slang, jargon, and humor.

  Slang, jargon, and humor are specific for a particular language and cannot be easily translated into another language.

- Avoid negative questions.

  Negative questions are often misunderstood by the user. When asking questions, ask them in a positive way.

Textual data code design: Application displays, printer file specifications, and user-created commands usually contain a large amount of constant text. Application displays, printer file specifications, and user-created commands also contain input and output fields such as headings, field prompts, instruction lines, and function key descriptions.

You can use different techniques to specify, store, and use constant text. You can use each technique for specific types of textual data components. Each technique has its advantages and disadvantages. The following topics show you how each technique works and describe which techniques you can use for various components:

- Early binding of messages
- Late binding of messages
- Direct coding as an unnamed output field
- Text stored in database files

Early message binding: Text can be stored externally from the source code in a separate message file but is bound into the object when it is created. This technique can be used for:

Display files
: Constants such as titles, instruction lines, option definitions, headings, field prompts, command key descriptions

Printer files
: Constants such as titles, headings, total line descriptions

User commands
: Prompt descriptions on the command definition statements

For device files (display and printer), the message is referred to by the Message Constant (MSGCON) keyword in the DDS source specifications.

For example:

```
A          line pos   MSGCON(length message-ID [*libl/]message-file-name)
                                ^
                           includes expansion space
```

For user commands, the message identifier *xxxnnnn* is specified on the PROMPT keyword instead of a literal. The message file is referred to on the Create Command (CRTCMD) command.

For example:

```
CMD      PROMPT(xxxnnnn)
```

The message file name *message-file-name* is in a source file referred to by the following command.

```
CRTCMD CMD(command-name)  PGM(library-name/program-name)  +
     PMTFILE([*libl/]message-file-name)
```

Before the object can be created, you must enter the message description into the specified message file. Enter the message description using the Add Message Description (ADDMSGD) command.

For example:

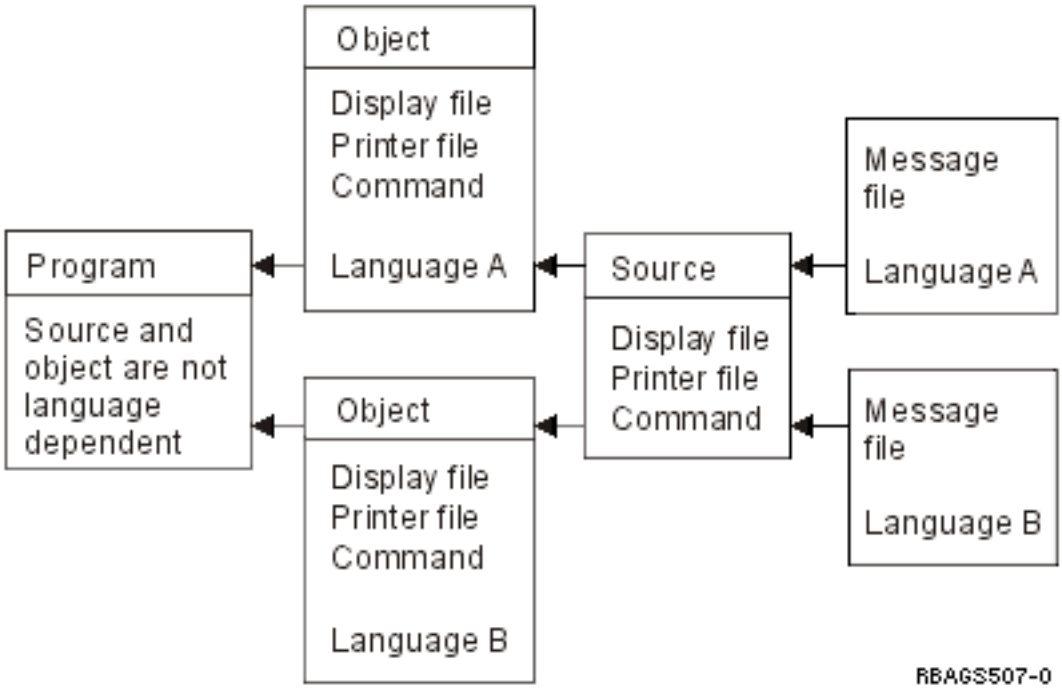
```
ADDMSGD    MSGID(xxxnnnn)  MSGF(library-name/message-file-name) +
MSG('Text                                          ')
```

where *xxxnnnn* is the message identifier.

This technique allows you to create any number of objects in different languages and to put them into different libraries using the same source code by just assigning another message file at object creation time.

The message file is needed only during the creation of the object. Consider specifying the appropriate length for different languages on the MSGCON keyword. Then make the length information available to the translator.

The following graphic shows how early message binding works:



At file creation time, you can choose the appropriate textual data of the language version you want to work with by setting up the library list with the specific library containing the textual data and the program library.

Late message binding: Text can be stored externally from the DDS source code in a message description and is bound only to the display format at run time.

This technique can be used for:

Display files only

Constants such as titles, instruction lines, option definitions, headings, field prompts, command key descriptions (MSGID keyword)

Default values on input fields (MSGID keyword)

Field validation specifications (CHKMSGID keyword)

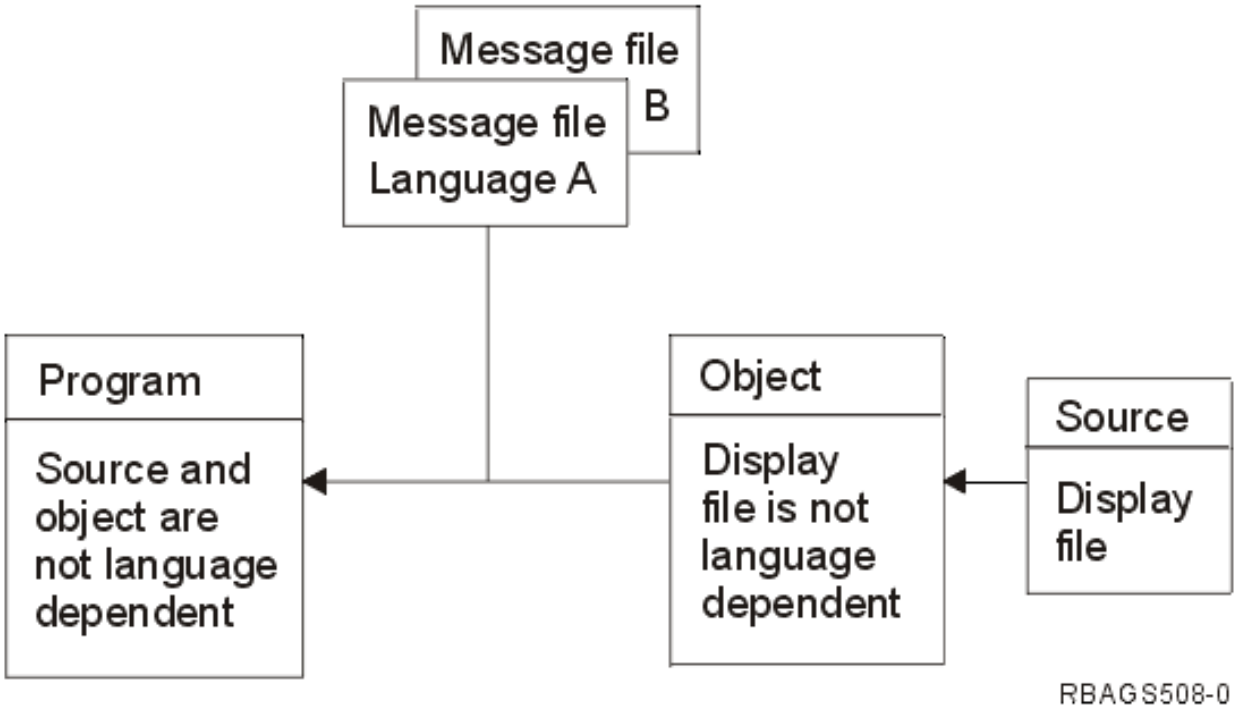
Error messages (ERRMSGID and SFLMSGID keywords)

In the DDS for the display file, the message is specified through the MSGID (Message Identifier) keyword. The message has to be entered into the specified message file using the ADDMSGD (Add Message Description) command.

For example:

```
A    FLD-name  length  line pos    MSGID(message-ID [*libl/]message-filename)
                  ^
          includes expansion space
ADDMSGD   MSGID(xxxnnnn) MSGF(library-name/message-file-name)  +
MSG('Text                                          ')
```

This technique allows you to create any number of message files in different languages and different libraries, with one DDS source code and display file object. During run time, you assign another message file by setting the library list accordingly. The following figure is an example.



Note: This technique requires the application to perform all editing based on the cultural convention.

Direct coding as an unnamed output field: The most common way to define constant text is to specify the text directly in the source code as a literal. While this method is the most common way to define constant text, it is the most difficult to translate. Avoid using this method whenever coding an application, even if the application is not planned for translation.

If you are coding an application that will not be translated, you may want to use this technique for:

Display files

Constants such as titles, instruction lines, option definitions, headings, field prompts, command key descriptions

Default values on input fields (DFT keyword)

Error messages (ERRMSG/SFLMSG keyword)

Printer files
Constants such as titles, headings, total line descriptions

User commands
Prompt descriptions on the command definition statements.

For device files, specify the text as an unnamed field, indicating the starting line and column and the constant text itself.

For example:

```
A              line pos       'Text . . . . . . . . . . . . . :  '
```

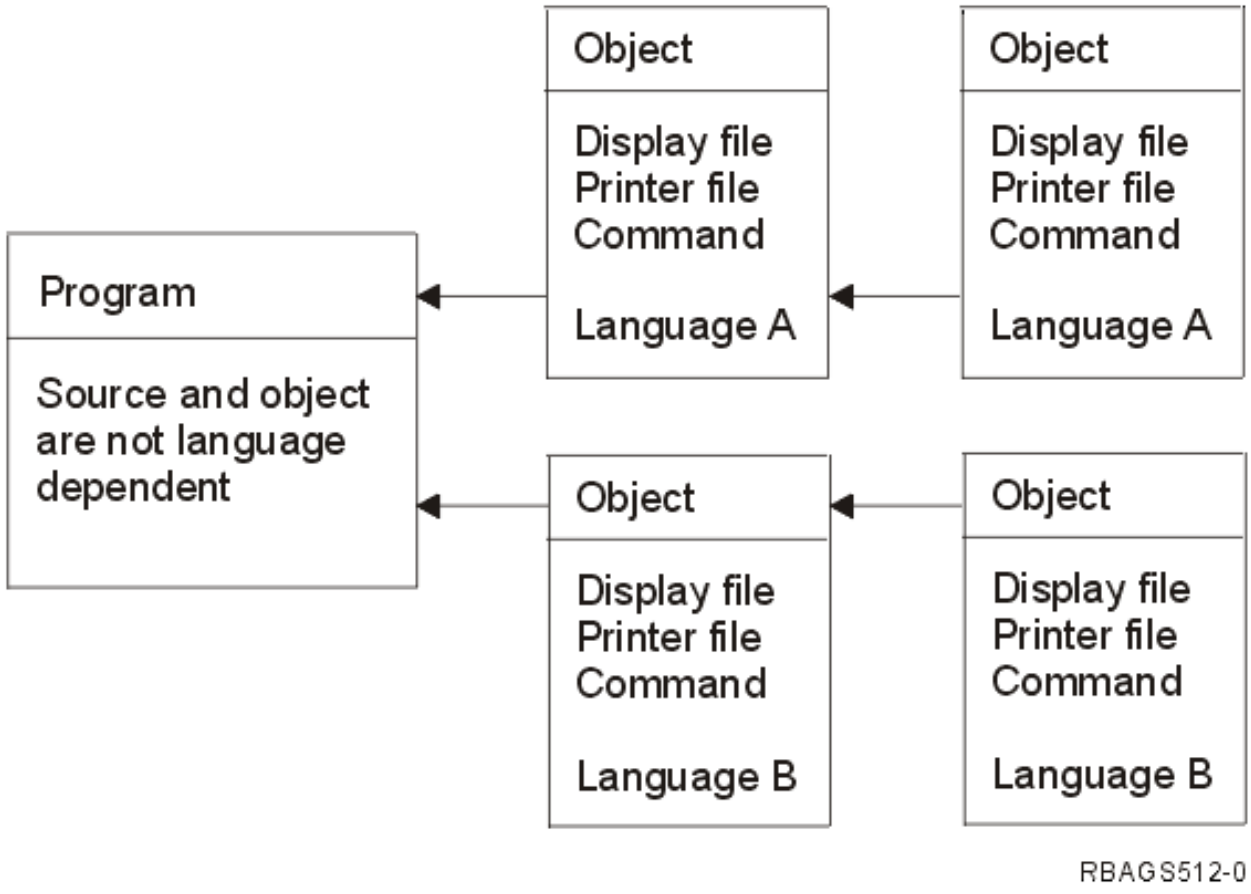
A similar rule applies to user-created commands. Define the text directly on the keywords of your command source statements.

For example:

```
CMD           PROMPT('    Command description       ')
```

When defining the text directly on the keywords, standardize the sizes of the different elements in a large literal, rather than specifying many small single ones as single words. This makes the source code more readable and more flexible for translation.

Consider that the space needed for explanation text can vary from language to language. To have enough room after translation, remember to reserve space initially. The source members need to be translated and the objects need to be created for different languages as shown in the following figure:



Each national language version has one set of programs, but can have multiple sets of source members and data objects. When the application is run, you can choose the appropriate textual data of the language version that you want to work with. This can be done if you set up the system part of the library list with the specific library that contains both the textual data and the program library.

Text stored in database files: Text can be stored externally from the source code in a database file, retrieved by the application program, and then moved to the display or print format at run time. Instead of coding constants on the DDS, you can specify output fields that can be filled by the program. Consider specifying the appropriate length for different languages on the output fields and making that available to the translator.

This technique can be used for:

Display files
All constant text
Default values on input fields
Error messages

Printer files
All constant text

Programs
All constants like compare values, scan characters, and tables.

This technique allows you to create any number of database files in different languages and different libraries, with only one DDS source code and display file object. During run time, you assign the corresponding database file by setting the library list accordingly.

Note: This technique requires the application to perform all editing based on the cultural convention.

User interface manager: The OS/400 user interface manager (UIM) is a part of the system that allows you to define panels and dialogs for your application. UIM provides the following support:

- A tag-based language for describing data and panels.
- A compiler to create panel group objects and menu objects by using the tag-based language.
- A set of application programming interfaces (APIs) to use as panel group objects to display and print panels.

The UIM also provides the following functions:

- Dialog commands for screen management
- Contextual online help
- Pop-up windows
- Menu bars
- Command line for entering CL commands
- Tailoring of the contents of a panel for different users or environments
- Fast paths through menu networks
- Double-byte character set (DBCS) languages
- Bidirectional (BIDI) language support

UIM supports common panel types such as menus, information displays, list displays, and entry displays. When all display types and interfaces are consistent, users adapt more quickly to new applications.

UIM applications can coexist with and share the requester display device with other open display files that are not under UIM control. However, a UIM panel and a DDS-defined record format cannot appear on the display at the same time. When a UIM panel either replaces a DDS panel or vice versa, the system suspends operations of one file or panel group and restores the display as needed.

The following provide more information for user interface manager:

- Online help design
- Index search tags
- Index search and DBCS

Online help design: You can define online help by using one of the following:

Panel groups
Objects into which user interface manager (UIM) source is entered.

Records
A set of DDS keywords contained in a source file member.

If the user interface manager is used for defining online help, the panel groups are defined either in place of DDS or in the display file. In either case, the encoding of the data to be displayed must be indicated by the CHRID value in the display file or the panel group.

A panel group is an object that can be used to contain help information. OS/400 uses *PNLGRP as an identifier for the object type that contains a collection of help information.

Guidelines: Online help

When defining online help information to be translated into national language versions, keep in mind the following about panel groups and records:

- Records do not have word processing available (functions such as spell check and word wrap though system APIs exist to provide spell checking).
- Various OS/400 messages and panel groups determine the national language conventions and translations. Not all countries have a national language version available for the OS/400 program. Not all national language versions are completely translated, with many parts still in English. The messages and panel groups that are not translated do not reflect the national language cultural conventions. See Command design for an example of a translated panel in which part of the panel has remained in English because not all parts of the NLV were translated.
- Allow for translation expansion.

Guidelines: DDS online help design

When multiple languages are installed on one system, the help documents are stored in different folders. The DDS source file needs to be copied, changed, and compiled again for each language on the system.

Index search tags: Help panel groups may contain index search modules. Index search supplements the help information that is provided for each display. To use the information in help panel groups for the index search function, you need to add the appropriate UIM tags to your help modules.

Users can access the index search function from any display help that specifies that the index search function is available.

The ISCH tag

The ISCH tag defines the title of a topic in the index and specifies the root words that serve as the link between the topic and the search words (synonyms) entered by the user. The tag appears immediately after the HELP tag to which it refers. There can only be one ISCH tag within a single help module.

For each ISCH tag, there can be several lines of root words, provided that the total number of root words is no more than 50. If more than one line of root words is used, ROOTS= must be repeated at the beginning of the second line and subsequent lines:

```
:PNLGRP.
:HELP name=entry1.
:ISCH ROOTS='root1 root2 root3 root4 root5'
 ROOTS='root6 root7 root8 root9 root10'
 ROOTS='root11 root12 root13 ... root50'.
Title of First Topic

This is the first index search module in this panel group.
:EHELP.
:EPNLGRP.
```

The root words on all lines must be enclosed in apostrophes and a period must be placed only at the end of the last line of root words. The topic title follows the period on the ISCH tag and may be placed on the line immediately following the period.

The ISCHSYN tag

The ISCHSYN tag defines the words (synonyms) that, if entered by a user, match a specific root word. If a word that is entered by a user is a synonym for a root word, then a match is found for each topic whose ISCH tag contains that root.

If you want a word that is used as a root word to be used as a synonym as well, you must include the word as a synonym on the ISCHSYN tag. For example:

```
:ISCHSYN ROOT='ocean'.ocean water sea
```

The synonyms for the ISCHSYN tag must be entered on one line, and at least one ISCHSYN tag must exist for each root word. If more than one line is needed, more ISCHSYN tags may be entered for the same root word.

UIM does not differentiate between synonyms entered in uppercase, lowercase, or mixed case. For this reason, it is not necessary to repeat synonyms to cover all the different cases.

You may use alphabetic or numeric characters for synonyms; however, the following characters (including their hexadecimal equivalents) are not allowed to be used as a synonym or part of a synonym:

- . (period)
- ((left parenthesis)
-) (right parenthesis)
- ; (semicolon)
- , (comma)
- ? (question mark)
- : (colon)

The ISCHSYN tags may be placed anywhere in the panel group, but to make maintenance and translation easier, place them all in one area (such as at the beginning of your panel group or in a panel group object that contains only ISCHSYN tags).

Example: ISCH and ISCHSYN usage

The following example shows some ISCHSYN tags and the ISCH tags that use them:

```
:PNLGRP.
:ISCHSYN ROOT='ocean'.ocean water sea
:ISCHSYN ROOT='lake'.lake water pond
:ISCHSYN ROOT='definition'.definition define description what
:ISCHSYN ROOT='definition'.summary concept information explanation
:HELP name='defocean'.
:ISCH ROOTS='definition ocean'.
Definition of ocean
```

```
An ocean is one of the five large bodies of salt water, which
together cover nearly three-fourths of the world.
:EHELP.
:HELP name='deflake'.
:ISCH ROOTS='definition lake'.
Definition of lake

A lake is a body of standing water that is enclosed by land.
:EHELP.
:EPNLGRP.
```

Index search and DBCS: The index search function can be used with either double-byte character support (DBCS) or single-byte character support (SBCS) data. When DBCS data is used, the device from which it is requested must be capable of entering and presenting the data in DBCS. The object that contains the index search data is marked as containing DBCS data. The system determines if the device is capable of handling the DBCS data.

When the data is being prepared for DBCS format and the index search function is used with that data, consider the following:

- When the index search data is prepared for a DBCS system, the synonyms entered on the ISCHSYN tag must be in double-byte character mode. That is, the first byte after the tag must be a shift-out character and the last byte of the data must be a shift-in character. The system does not convert data on the ISCHSYN tag to double-byte character data.
- Words must be separated by a single-byte blank. From 1 to 19 double-byte characters may be combined to form a word. Intervening shift-out and shift-in characters are allowed, but are ignored by index search.
- The words that are used to link the ISCH and ISCHSYN tags (the ROOTS attribute of the ISCH tag and the ROOT attribute of ISCHSYN tag) must be identical and should not be entered in DBCS.
- Search words can be entered in either single-byte mode or double-byte mode. Single-byte blanks can be entered to separate the words.

When the search words are shown on the screen, the double-byte character representation (the character that was actually used in the search) is shown. Special processing takes place so that index search is not case sensitive. The search words from the ISCHSYN tag are converted to uppercase using a conversion table for the code page that is specified with the TXTCHRID attribute of the PNLGRP tag. If the search words are DBCS, they are not converted to uppercase. Shift-out and shift-in characters are treated as blanks during parsing; leading and trailing blanks are removed. All SBCS words are converted to uppercase using a conversion table for the code page of the device description.

Program message design: On OS/400, a message can be predefined or immediate. Consider the following when designing and coding your messages:

- Do not use immediate messages. They are created by the sender or program at the time they are sent and are not stored in a message file. Therefore, they cannot be translated by the translator.
- Use predefined message descriptions that can both:
  - Exist outside of the program that uses them.
  - Be stored in a message file.
- Do not specify the maximum size for a message file. When the message file becomes full, you cannot change the size of the message file. You need to create another message file and add the message description again.

  Use the Create Message File (CRTMSGF) command to create a message file to hold the predefined message description. The contents of the predefined message description can be put into a message file by the Add Message Description (ADDMSGD) command. For details, refer to the Control Language information.

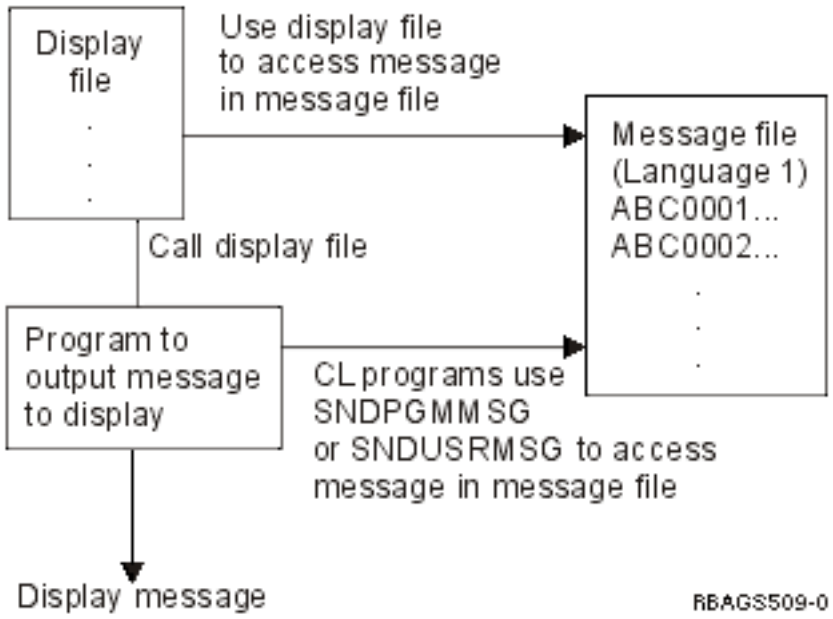
- Use substitution variables with care. Different languages have different orders for substitution variables. For example, in the English message:

  ```
  File &1 in Library &2 not found.
  ```

  &1; and &2; are the substitution variables. Those substitution variables may appear in different positions for different languages.
- Make your design and coding able to understand a reply code for different languages. For example,

  ```
  English     Y = Yes
  Danish      J = Ja (means Yes)
  ```

The following figure shows the creation of different NLV messages from message files.



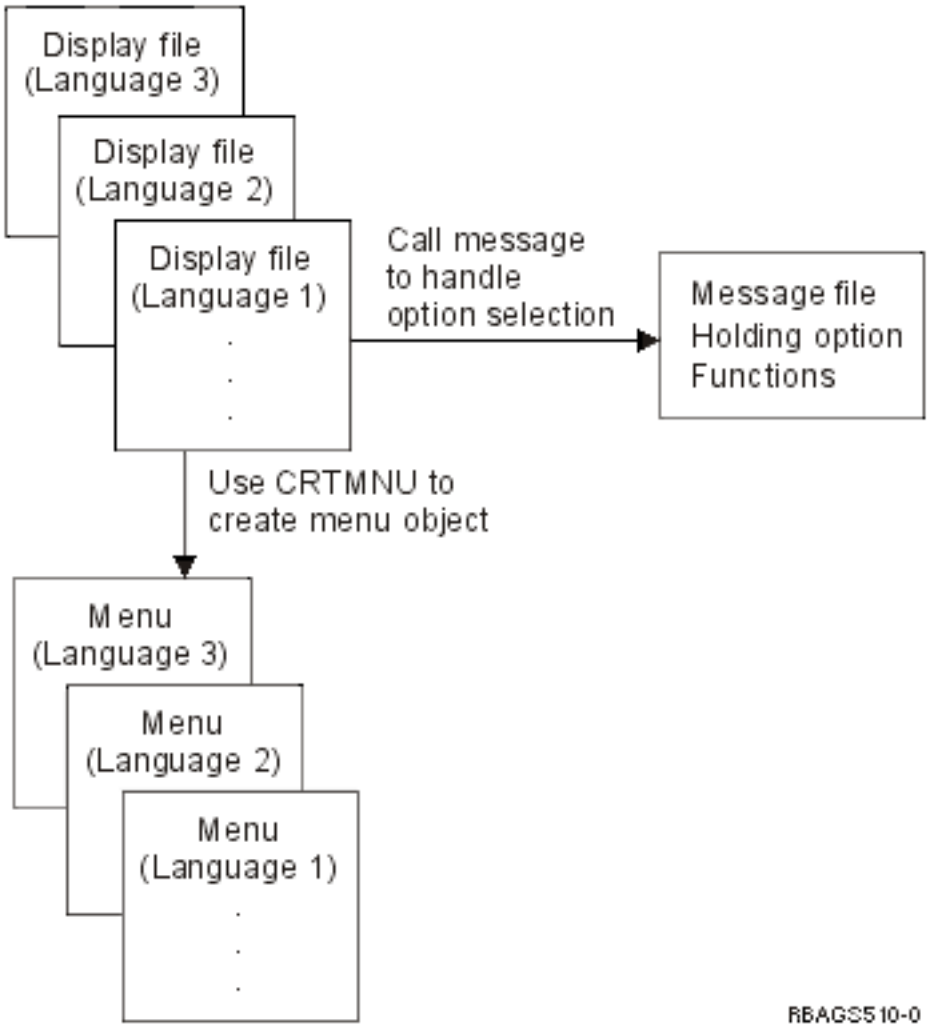
A program can directly access the message file for program messages, or it can indirectly access the message file through display files for program messages. For more information on message files, see CCSID support for messages.

Menu design: You can define your own menus on OS/400. There are three types of user-defined menus: display file menus, UIM (reference) menus, and program menus.

To use an application system, users have to deal with a lot of menus and displays. When an application is being translated from one language to another, a large portion of the literal text to be translated comes from menus.

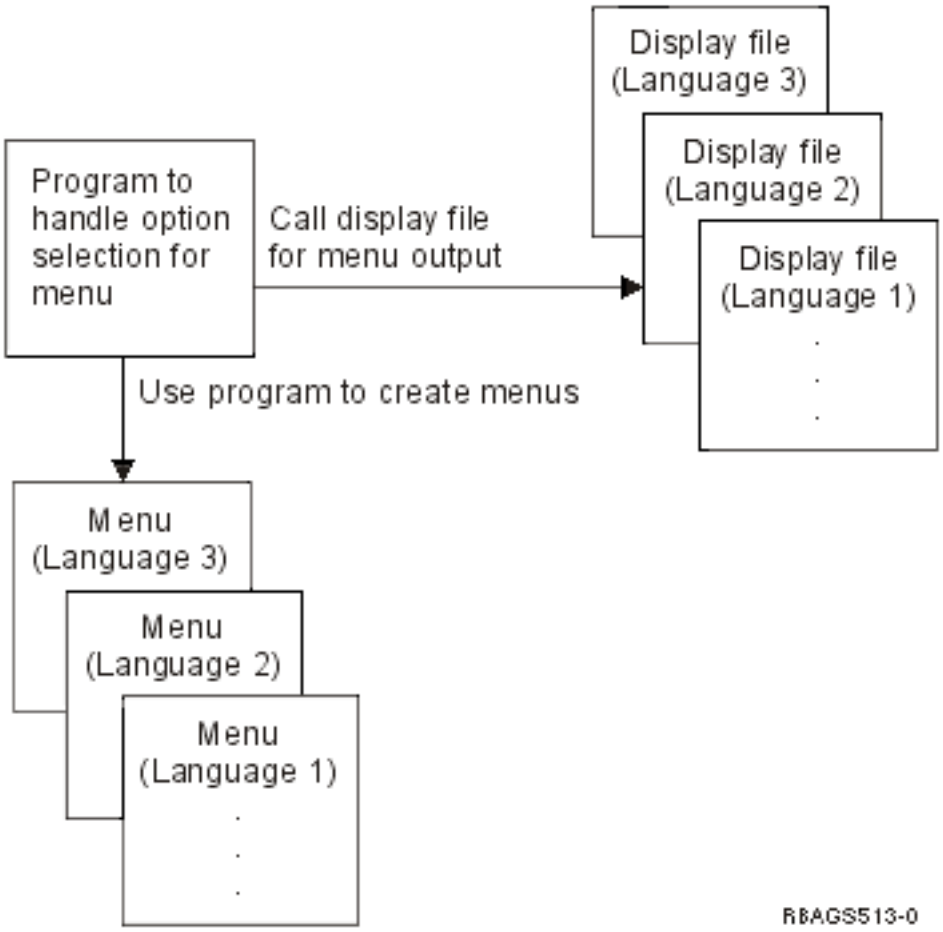
Display file menu

A display file menu uses a display defined by DDS to present a menu format. The menu functions are controlled by a menu object that contains the commands used to run each of the menu options. The following figure shows how display file menus are created for different national language versions.



Program menu

A program menu uses programs to present the menu format (defined by DDS) and to provide functions necessary to run the menu options. The following figure shows how program menus are created in different national language versions.



Menu Translation

To allow for easy translation into national language versions of your menus:

- Keep the literal text of menus external by holding the constant text as externally defined message descriptions in a message file and by incorporating the text into a menu file when the program is run.
- Be aware of the expansion space needed when a menu is translated from one language to the next. Leave space for translation expansion when you design your menus.
- Be aware of cultural conventions when date, time, or edited fields are displayed on the menu.
- Use numerals 0 through 9, instead of uppercase and lowercase English letters (A through Z), as the option fields for selection. Numeric characters are more standard among different languages.

Command design: The OS/400 program allows users to define and create their own commands. To create a command, you must first define the command through command definition statements. Then use the Create Command (CRTCMD) command to process the command definition statements to create the command definition object.

When defining and creating a command, take into consideration the following:

- Use help panel groups to provide online help information for the command. See “User interfaces” on page 14 for information on national language version help panels.
- Use message identifiers instead of literal text for the PROMPT keyword on the CL CMD, PARM, ELEM, and QUAL command definition statements.

- Translate the text that is displayed to the right of the prompt line of each parameter on the prompt display. This text is specified by the CHOICE parameter of the PARM command definition statements, so the appearance of the command prompt display will maintain its coherency.
- Compile command-prompt text into separate command definition object versions for each national language. Use the Change System Library List (CHGSYSLIBL) command before creating the command to get the national language version prompt text from the correct national language version library.
- The function keys of the command prompt display are provided by the OS/400 program. If the NLV of the OS/400 program is different from the NLV of the command, two different languages would appear on the command prompt display. For example, when translating an English display into German, both the English and German would appear on the command prompt display.

  For additional information on creating and defining commands, see Control language.

Cultural-dependent design: Different countries have different standards that must be taken into account when developing an NLS-enabled application. This culturally sensitive information must be placed outside the program the same way as the textual data is handled.

Many languages have characters (such as common-usage vowels essential to the correct spelling of a word) outside of the A-Z alphabet that must be considered for collating purposes.

Through system values, the system supplies linguistic support, cultural support, and the ordering of data. For a list of the default system values for each national language version, see Default system values.

The following topics address the various attributes that should be considered when designing culturally dependent applications:

- Database file attributes
- Job attributes
- Program attributes
- Information in message CPX8416
- Date formats
- Date separators
- Editing date presentation
- Time formats
- Time separators
- Editing time presentation
- Decimal format
- NLS sort sequence

Database file attributes: Culture-dependent database attributes are the following:

- Coded character set identifier (CCSID)
- Sort sequence (SRTSEQ)
- Language identifier (LANGID)

The CCSID attribute applies only to physical files. The SRTSEQ and LANGID attributes can be used with both physical files and logical files. A logical file can have a CCSID value only when it has taken the CCSID from the physical file. The database attributes are stored with the data. They are static in the sense that they cannot be dynamically altered by the process of accessing the data.

Job attributes: Culture-dependent job attributes are the following:

- Coded character set identifier (CCSID)
- Sort sequence (SRTSEQ)
- Language identifier (LANGID)

- Country or region identifier (CNTRYID)
- Date format (DATFMT)
- Date separator (DATSEP)
- Decimal format (DECFMT)
- Time separator (TIMSEP)

The default values for CCSID, SRTSEQ, LANGID, and CNTRYID attributes are set from the user profile when the job starts. The values for CCSID, DATFMT, DATSEP, DECFMT, SRTSEQ, and TIMESEP can be set from the LOCALE and SETJOBATR attributes associated with the user profile. When you use the Change Job (CHGJOB) command, you can override the values specified for any of the listed job attributes.

Program attributes: The SRTSEQ and LANGID parameters can also be specified as program attributes belonging to a *PGM object type. The LANGID parameter is used together with the SRTSEQ parameter only when the SRTSEQ value is set to *LANIDUNQ or *LANGIDSHR. Otherwise, the LANGID parameter is not used.

If a program explicitly refers to a sort sequence or a language identifier, then those attributes stored in the program object take effect. The *JOBRUN value for these parameters is used to refer to the attributes of the job running the program. *JOBRUN makes it possible to use a single set of programs processing data according to different sort sequences. The *JOBRUN value affects only the processing of data, however, not the retrieval sequence of data. The retrieval sequence is determined by the database attributes. To retrieve data in a sort sequence different than what is defined in the database, use logical files that are built separately.

Information in message CPX8416: If your application will be translated into other languages, use the message CPX8416 from the QCPFMSG message file to get the correct setting for some cultural values for the other languages. The message exists for your primary language and all installed secondary language libraries. The system message contains these values:

- Code page and character set
- Currency symbol
- Date format
- Date separator
- Decimal format
- Leap year adjustment
- Coded character set identifier
- Time separator
- Language identifier
- Country or region identifier

Culture-dependent fields in the panel or display should not contain hard-coded values. These fields should be defined with the maximum length permitted for the field on the display.

If your application is to support users in languages other than the primary language, the callable routines should use the CPX8416 message values. A callable routine uses the cultural values for the primary language to determine the contents of the field (for example, date format) and places these values on the display. NLS system values maintained in message CPX8416 determine the format of the cultural values appearing in the culture-dependent fields.

Your application can use the details from the system message.

The following table shows the layout for message CPX8416. This example shows the values in the text column using the English uppercase and lowercase NLV (feature 2924).

	Field	Start	Length	Justify
Description Value	QCHRID 697 37	0001 0012	10 21	L L
Description Value	QCURSYM $	0034 0045	10 01	L L
Description Value	QDATFMT MDY	0047 0058	10 03	L L
Description Value	QDATSEP /	0062 0073	10 01	L L
Description Value	QDECFMT	0075 0086	10 01	L L
Description Value	QLEAPADJ 0	0088 0099	10 01	L L
Description Value	QCCSID 37	0101 0112	10 05	L L
Description Value	QTIMSEP :	0118 0129	10 01	L L
Description Value	QLANGID ENU	0131 0142	10 03	L L
Description Value	QCNTRYID US	0146 0157	10 02	L L
Description Value	QIGCCDEFNT *NONE	0160 0171	10 21	L L

Date formats: There is no worldwide standard for the presentation of dates. Therefore, the date format should always be stored externally as part of the textual data. The valid date formats on OS/400 are:

- *MDY (Month, day, year)
- *DMY (Day, month, year)
- *YMD (Year, Month, Day)
- *JUL (yy/ddd)
- *ISO (YYYY-MM-DD)
- *USA (MM/DD/YYYY)
- *EUR (DD.MM.YYYY)
- *JIS (YYYY-MM-DD)

Note: Some OS/400 functions do not support all of the date formats shown above.

In database files, dates can be stored as:

- Normal numeric data fields
- SAA(R) date data-types

When you store dates as numeric data, your application needs to specify the format in which it is stored and presented.

When you store dates as data type DATE (L), you can specify the format with the DDS keyword DATFMT on the database file. The date is shown in this predefined format as character data, including the date separators.

If date sorting and other processing is needed, the date should be stored in *ISO format (YYYY-MM-DD) and converted to another format during the input and output operations. Write a high-level language routine to convert dates.

Date separators: The valid date separators are:

- / (slash)
- - (dash)
- . (period)
- , (comma)
- (blank)

The date separator for presentation should always be stored externally as part of the textual data.

When you use decimal fields for dates, not only does your application have to specify the format, but it also must handle the date separators during the input operation and presentation.

When you use date-type fields, the date separators are always included in the date. To change the date separator, you can write a high-level language routine to convert dates.

Editing date presentation: You need to handle the presentation of dates on display and printer files differently, depending on how they are stored:

- As a normal decimal data field

  Your application program has responsibility for the way the date is entered, stored, and presented. The application must check to see that the date is entered in the right format, remove any date separators, convert the date to another format when necessary, and edit it on the display file or printer file.

  The DDS keyword DATE is used as an output-only field. DATE uses the job attributes DATE, DATFMT, and DATSEP. You can edit DATE using the edit code keyword, EDTCDE, for 6- and 8-digit date fields.

  Editing with EDTCDE includes the following changes to the appearance of displayed fields, depending on which edit code is specified:

  – Leading zeros are suppressed.
  – Zero values can be displayed as zero or blanks.
  – The field can be further edited using a user-defined edit code.

  For all other types of fields using the EDTCDE Y keyword, the program has to specify the format, and the system uses the date separator of the job that created the device file. The date separator is integrated in the object, and you are not able to change it dynamically at run time.
- As an SAA data type DATE (L) field

  The DDS date format (DATFMT) keyword allows you to specify different date formats, including default date separators, at the database field level. For the *MDY, *DMY, *YMD, and *JUL parameters, the default date separator can be changed with the date separator (DATSEP) keyword. The *ISO, *USA, *EUR, and *JIS values have a fixed separator, and the DATSEP keyword is not allowed with these values. The DATFMT and DATSEP keywords allow you to specify the format and editing characters for storing date fields. The date is shown as a character string, including the separators.

  Any format conversion between the date input and the format the database asks for can be done by:

  – Application program routines
  – Field mapping through logical files that define different date formats and separators

  For example, you can provide a date conversion that is dependent on the actual job attributes by using the following CL program:

```
PGM      PARM(&fromfmt &fromfld  &tofld  );
DCL      VAR(&fromfmt); TYPE(*CHAR)  LEN(4)
DCL      VAR(&fromfld); TYPE(*CHAR)  LEN(10)
```

```
DCL      VAR(&tofld);  TYPE(*CHAR)  LEN(10)
CVTDAT   DATE(&fromfld); TOVAR(&tofld);
FROMFMT(&fromfmt); TOFMT(*JOB)  TOSEP(*JOB)
ENDPGM
```

Your application program has to pass the format of the date you want to convert and the date itself to the CL program. The CL program assumes that the job attributes represent the way the user expects to see date fields edited. It retrieves these values and does the conversion, conforming to these values, and passes back the date in that way. The *ISO, *USA, *EUR, and *JIS values have a fixed separator that cannot be changed. If the TOFMT parameter contains one of these values, the TOSEP value is ignored.

Time formats: The time formats supported on OS/400 are:

- *HMS (hh:mm:ss)
- *ISO (hh.mm.ss)
- *USA (hh:mm AM or hh:mm PM)
- *EUR (hh.mm.ss)
- *JIS (hh:mm:ss)

The system value QTIME has one format (hhmmss). The time separator value is determined by the QTIMSEP system value.

The time format for presentation should always be stored externally as part of the textual data.

In database files, times can be stored as:

- Normal numeric data fields
- SAA time data-types

When you store the time as numeric data, your application needs to specify the format in which it is stored and presented.

When you store the time as data type TIME (T), you can specify the format with the DDS keyword TIMFMT on the database file. The time is sorted in this predefined format as character data, including the time separators. To convert time fields from one format to another, write a CL program or high-level language routine to do the conversion.

Time separators: The valid time separator characters on OS/400 are:

- : (colon)
- . (period)
- (blank)
- , (comma)

The time separator for presentation should always be stored externally as part of the textual data.

When you use decimal-data fields for time fields, your application needs to specify the format and time separators on the input and presentation operations.

When you use time-type fields, the time separators are always included in the time field. To change the time separators, write a CL program or high-level language routine to do the conversion.

Editing time presentation: You need to handle the presentation of times on display files and printer files differently, depending on the way they are stored:

- As a decimal data field

Your application program has responsibility for the way the value is entered, stored, and presented. The program must check for the correct format, eliminate the time separators, convert the time to another format when necessary, and edit it on the display file or printer file.

The editing can be done by specifying the edit word (EDTWRD) for the field. The TIME keyword is an output-only field. Both the edit word and TIME keyword use the information available at creation time. The time separators are integrated in the device file object.

Both ways force you to have different copies of the source and objects for different editing requirements.

- As an SAA data type TIME (T) field

  The OS/400 program allows you to specify different time formats and time separators on the database file level. The TIME keywords allow you to specify the format and editing characters for storing time fields. The time type field is shown as a character string that includes the separators.

  As an SAA data type, you can specify such time fields as normal character fields on the display file or printer file. On an input operation, your program has to check entered values for the correct format and separators and move them over to the database field. On an output operation, you just move the character string from the database file field to the device file field, including the separators. Any format conversion between the input and output format and the format that the database asks for can be done either by:

  – Application program routines
  – Field mapping through logical files that define different time format and separators

Decimal formats:

You can change the decimal format with the QDECFMT system value to reflect the way decimals are presented for your country or location.

Sort sequences: Sort sequence is supported on OS/400. You can order your data according to culture-dependant requirements for specific applications by using one of the following options:

- Hexadecimal sorting (sort sequence tables not used). This is the default.
- A user-supplied or system-supplied shared-weight sort sequence table or unique-weight sort sequence table, determined by the SRTSEQ parameter.

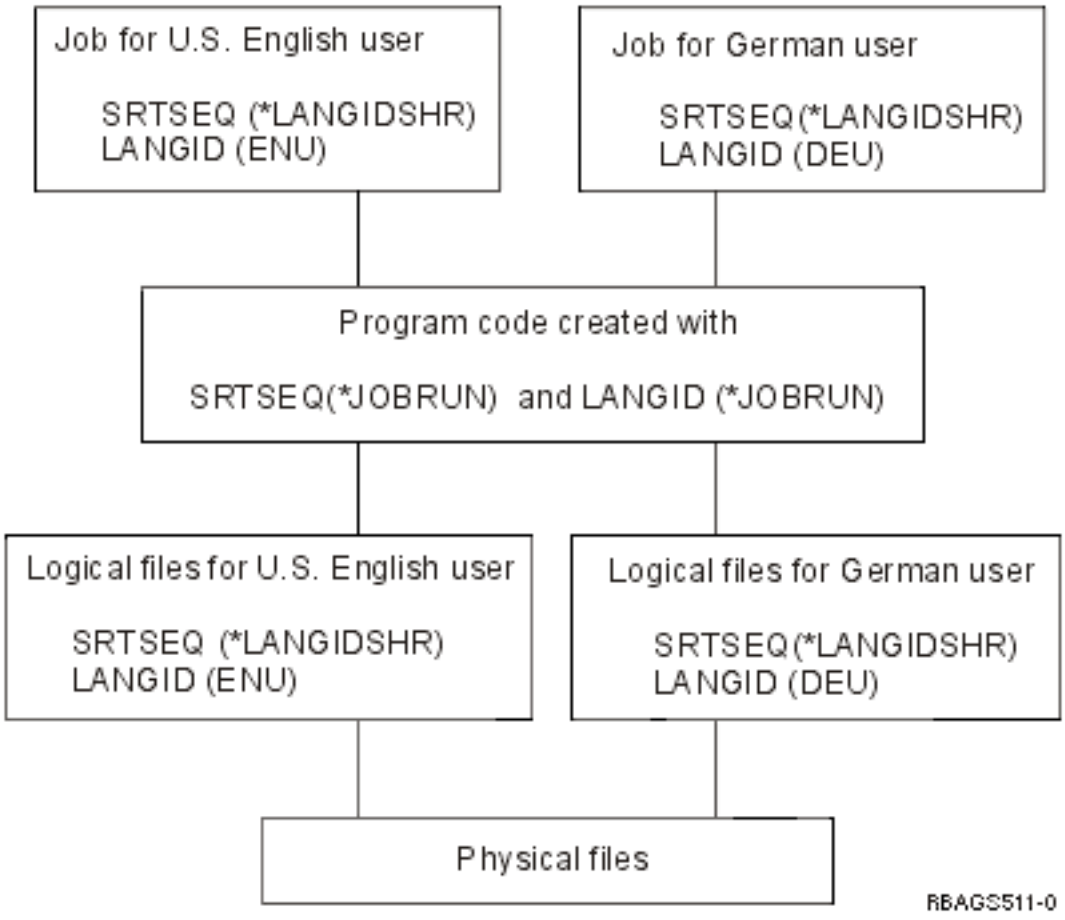
The following example shows how to use one DDS source file to create database files with different sort sequences. The following steps can be performed:

```
CRTxF     FILE(*CURLIB/NAME)
          SRTSEQ(*JOB)
          LANGID(*JOB)
```

You can then change the job attributes to create files with different sort sequences.

The CL program and high-level language programs can be created by specifying either early binding or late binding of a sort sequence. With early binding of a sort sequence, the sort sequence table to be used is determined at compile time. With late binding of a sort sequence, the sort sequence table to be used is determined at run time.

Late binding makes it possible to use one set of programs in different national language environments. The following figure illustrates using different sort sequences for different jobs with one set of physical files and program code. The sort sequence table defined for the job and used by the program should be the same as (or compatible with) the sort sequence table assigned to the logical files accessed through the library list.



Design for running with different sort sequences:

If your program is expected to run with different sort sequences, consider the following:

- Presenting the data in different order.
- Processing different records.

  Specifying selection criteria such as less than or greater than can result in selecting different records. The selection criteria equal to can result in selecting a different number of records when the shared-weight sort sequence table is used.
- Processing of a conditional branch may be different.

Note: System lists (such as the output from the WRKOBJ command) are not affected by sort sequence support.

You can use the DDS file-level keyword alternate sequence (ALTSEQ) to specify the sequencing table and the library in which it is contained. The system-supplied sort sequence tables with shared and unique weight can be used for defining the alternative collating sequence.

The alternative collating sequence table is inserted into the file at compile time and is not needed at run time. You can have different files containing different collating sequences using one set of DDS.

Note: The alternative collating sequence defined in your database files must also be defined in your application programs; otherwise, you may get unexpected results.

The DDS keyword ALTSEQ, provides limited support for sequencing, it has no effect on select/omit logic. The ALTSEQ keyword can only be used with the SRTSEQ(*SRC) parameter on the CRTPF and CRTLF commands.

For more information

See Character graphic (data) sort implementation for more information about sort sequences.

Display file design: Application panels usually consist of the following major elements:

- Constant text strings
- Input and output fields
- Field editing specifications
- Cursor positioning specifications
- Default values for input fields
- Field validation specifications
- Error messages

You can handle these either as a program-described or an externally described file using DDS. The information found in this topic is based on the externally described technique using DDS.

Constant text strings: Because different languages have different space needs for the same expression, design your panels with this in mind. Do not place many fields on the same line, except for a list panel that has column headings instead of field prompts. Do not overload the panels with information. Choose one of the techniques described under “Textual data code design” on page 18 to make your panels.

Input and output fields: Consider defining your fields according to the needs of the different languages, countries, cultures, currencies, and laws that you want to address with your application. For example, assume you want to store Italian lira and Japanese yen in the same field as United States dollars. You must size the field to accommodate the higher number of digits needed for Italian lira.

Field editing specifications: For the edit specification of your numeric, date, and time fields, consider the different cultural conventions of the users you want to address. Do not code the format and editing instructions in your application program in a way that requires program modification when another convention is needed. Refer to Cultural-dependent design for more information.

Cursor positioning specifications: Do not specify cursor positioning values to fixed locations on the screen, because different languages have different space requirements. When you work with different display files, you can adjust them with the translation process. When you need to work with field-independent cursor locations, store the positional information outside of your code and retrieve the variable values for the keyword within your program.

For example:

```
A     record-name      CSRLOC(field-name-1 field-name-2)
```

Cursor positioning on the field level is more useful in an NLS environment. For normal records, this is done by specifying the DSPATR(PC) keyword on a specific field. For subfiles, the cursor can be positioned using SFLRCDNBR(CURSOR) keyword on a special positioning field. In addition, the subfile record number must be stored in that field before the format is written.

For example:

```
A     field-name      4S 0B  line pos  SFLRCDNBR(CURSOR)
```

Note: The name of the record and field where the cursor is positioned, the subfile relative record number, and subfile fold/truncate indicator can be returned to your application program. This function is provided by hidden fields on the DDS keywords RTNCSRLOC, SFLCSRRRN, and SFLMODE.

Input field default values: There are three different ways to put default values into the input fields of your display, which the user can override with his own data:

- Getting information from program

  Never hard code the values as a literal if they are language or culture-dependent values. Use values you can get from the system-provided information, such as system or job date, or get the values from a data object, such as a database file or data area from outside of the program.
- Using DDS keywords DFT (Default) or DFTVAL (Default Value)

  Specify the default input value directly on the DDS after the keyword. The DDS keyword DFT is for input-only (I) fields. For output-only (O) or input-output (B) fields, use the keyword DFTVAL.

  For example:

```
A     field-name   length type I   line pos  DFT('default    ')
or
A     field-name   length type O/B line pos  DFTVAL('default value  ')
```

- Using DDS keyword MSGID (Message Identification)

  Using the Message Identification (MSGID) keyword allows you to retrieve the content of a specified message description when the program is run and to put that value as a default in your display file field. The field must be input-output capable (B) for you to use this technique.

  For example:

```
A   field-name  length type B line pos  MSGID(message-id [*libl/message-file)
```

  This allows you to use different message files for each national language version by setting the library list accordingly when the program is run.

Field validation specifications: The following DDS keywords provide validation checks on input-capable fields on your display:

- RANGE (Range checking)
- VALUES (Values checking)
- CMP and COMP (Comparison)
- CHECK (Check validity, keyboard control and cursor control)

Using the DDS keywords with any hard-coded values that are language, country, or culture-dependent makes duplication and modification of the DDS and the application program necessary.

Example: Validation checks

An example of field validation checks on input-capable fields on your display using the DDS keywords VALUES, COMP, and CHECK follows:

```
A     field-name   length type usage line pos  VALUES('Y' 'N')
or
A     field-name   length type usage line pos  COMP(EQ 'US$')
or
A     field-name   length type usage line pos  CHECK(M10 or M11)
(Modulus checking)
or
A     field-name   length type usage line pos  CHECK(RL)
(Right-to-left support)
```

Validation checks are provided according to the sort sequence defined for the display file at creation time. You can use the same DDS source file to create objects for different languages. For example, the following command creates a display object tagged with the Latin 1 sort sequence table:

```
CRTDSPF FILE(name) SRTSEQ(*LANGIDSHR) LANGID(DEU)
```

The following specification:

```
A     field-name   length type usage line pos  COMP(EQ 'a')
```

accepts all lowercase, uppercase, and accented characters, as defined by the shared-weight in the Latin 1 sort sequence.

In addition, note that all the checks specified using those DDS keywords are done by the data management function of the OS/400 program. Any error message caused by wrong input or handling by the user appears in the language of the OS/400 program. This could be the primary language or a secondary language, depending how the library list of the job is set up.

You can override this when you use the additional DDS keyword CHKMSGID (Check Message Identifier). This keyword allows you to specify your own customized messages and message file to be used by the checking routines of the OS/400 program.

For example:

```
A     field-name   length type usage RANGE(1 999)
A                                    CHKMSGID(USR1234 [*libl/]APPMSGF [&MSGFLD1])
A     MSGFLD1      length type   P   TEXT('Message data field')

and

ADDMSGD   MSGID(USR1234) MSGF(APPTXDENU/APPMSGF)
          MSG('Value &1; is out of range 1 to 999')
and

ADDMSGD   MSGID(USR1234) MSGF(APPTXDDEU/APPMSGF)
          MSG('Wert &1; ist ausserhalb des gltigen Bereichs 1 bis 999')
```

To use different message files of different library names, do not specify a fixed library name. You can use a message file for different languages by setting the library list when you run the program.

Error messages: There are two ways to provide error messages on a display file:

- Specifying text as constant on ERRMSG or SLFMSG keywords

  Specify the text directly as a constant on the DDS keyword. When you want to have more than one language, you have to duplicate the DDS source code and translate constants within the DDS specifications. You can then create a separate display file object for each language.
- Using predefined messages on ERRMSGID or SLFMSGID keyword

  When using predefined messages instead of constants, you need to have multiple display files.

  Instead of using different display files, exchange only the used message file by setting the library according to the language that you want to use.

  For example:

  ```
  A     field-name   length type usage EDTCDE(x)
  A 61                                 ERRMSGID(USR3456 [*libl/]APPMSGF [&MSGFLD2])
  A     MSGFLD2      length type   P   TEXT('Message data field')

  and

  ADDMSGD   MSGID(USR3456) MSGF(APPTXDENU/APPMSGF)
            MSG('Delivery date &1; is earlier than production end date &2')
  and

  ADDMSGD   MSGID(USR3456) MSGF(APPTXDDEU/APPMSGF)
  ```

```
          MSG('Lieferdatum &1; ist . . .')
.
.
.
```

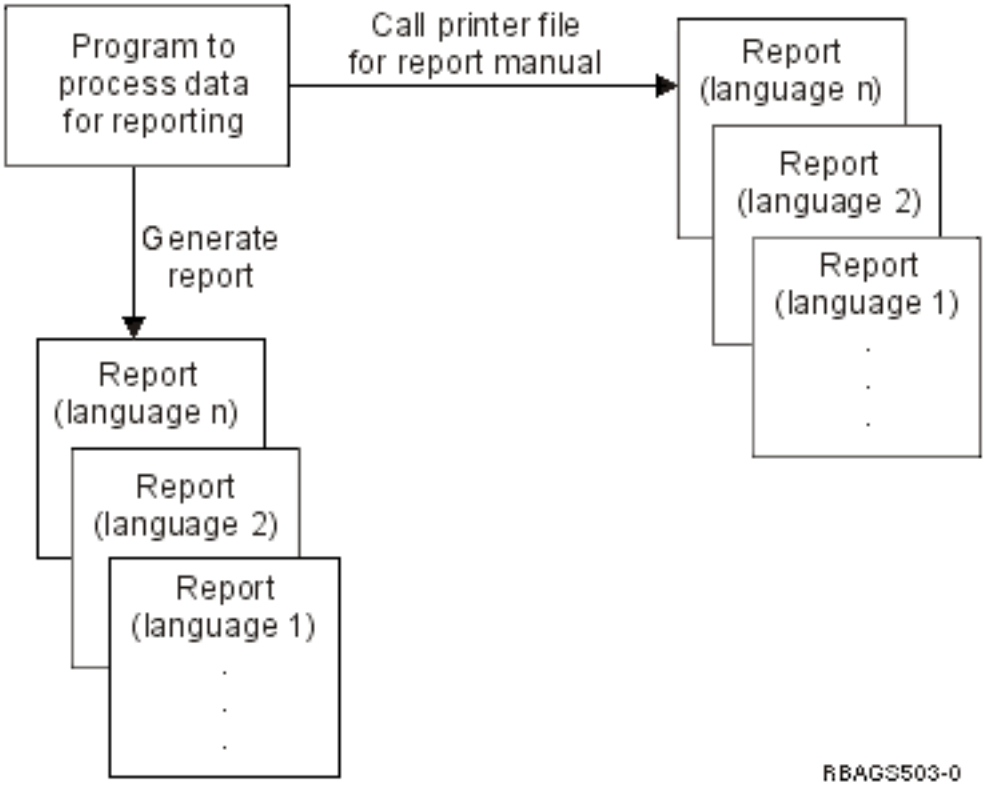
Printer file design and translation: The types of printer files are:

- Program-described printer files

  Program-described files rely on the high-level language program to define records and fields to be printed.
- Externally described printer files

  Externally described printer files use DDS rather than the high-level language to define records and fields to be printed.

The following figure shows how externally described printer files are used in creating reports for a different national language version.



Printer file translation:

When designing printer files to be translated into a national language version:

- Use externally described printer files to define records and fields to be printed. Avoid using program-described printer files. Program-described printer files are described inside the high-level language program. Translators trying to translate text imbedded within the program can mistakenly translate literals that are within your program.
- Print data in one national graphic character set on devices that support the corresponding character sets and code pages. Not all printers support all CHRID parameters.

- Use the MSGCON keyword to access the constant text described in the message file. A printer file does not have the MSGID keyword. However, the techniques of direct coding as unnamed output field (literal) and storing text in a database file can be used to specify the constant text in a printer file. See “Textual data code design” on page 18.
- Take culture conventions into consideration when bar codes are being described in the printer file. Different countries have different standards for bar codes.
- When entering data, consider these parameters on the Create Printer File (CRTPRTF) command.
  - PAGESIZE (page size)

    Different countries have different page-size standards.
  - OVRFLW (overflow line number)

    The overflow line number must be less than or equal to the page length.
  - CHRID (character set and code page)

    If the CHRID parameter of the printer file is set to *DEVD, the printer uses the character identifier that was set on the control panel or specified in the device description.

    If the CHRID parameter of the printer file is set to a specific value, this value determines the code page and character set used to print the data. For externally described printer files, the CHRID parameter is used only for fields that also have the CHRID DDS keyword specified. For all other fields, the code page and character set used is the same as if *DEVD was specified.

    If the CHRID parameter of the printer file is set to *JOBCCSID, constant text from an externally described printer file is converted to the CCSID of the job. The printer data stream is tagged with the CHRID taken from the job CCSID, using this CHRID value to print the data. When using the *JOBCCSID value on the CHRID parameter, the CHRID DDS keyword is ignored.

    Note: All code pages and character sets cannot be handled by all printers.

Source file design: Database source files are implicitly assigned the CCSID of the job when they are created, unless they have been explicitly assigned a CCSID value using the CCSID parameter on the Create Physical File (CRTPF) or Create Source Physical File (CRTSRCPF) command. If the job CCSID is 65535, the job default CCSID (DFTCCSID) is used as the implicitly assigned CCSID. The job default CCSID is determined by the system language identifier value and the job DBCS-capable indicator.

Character data representation architecture (CDRA) design: To enable your application for a multilingual environment, consider the following:

- Avoid coding CCSID values directly in your DDS for physical files. When creating different physical files for different languages, change the CCSID for your job (using the CHGJOB command). Only one set of DDS source code needs to be maintained.

  Conversions between all CCSIDs may not make sense in all cases. For example, if you access a Greek database with a CCSID of 00875 from a German display station with a job CCSID of 00273, you see garbled data on your display.

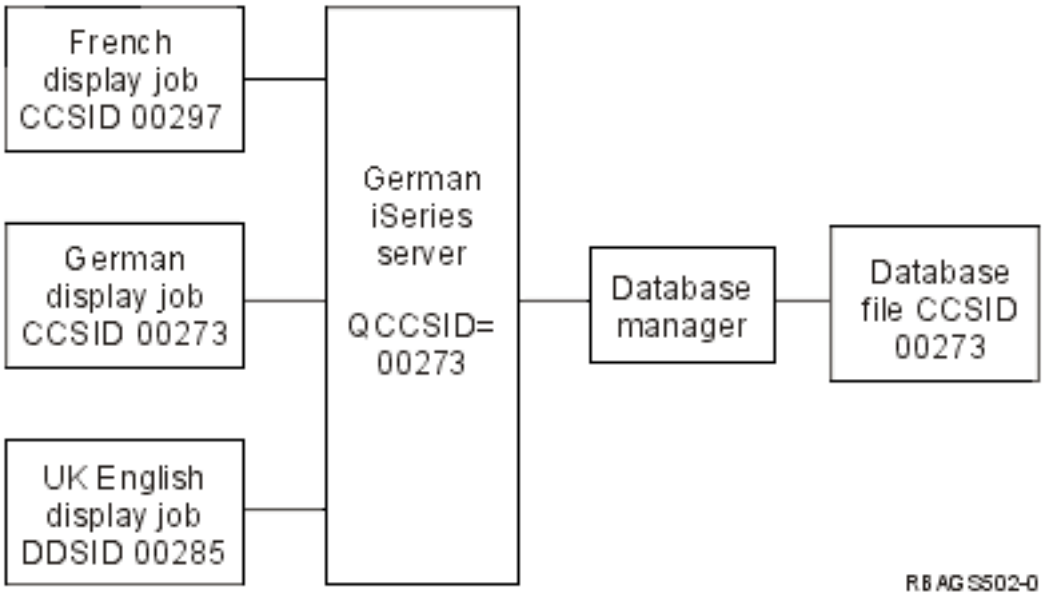
  Countries outside the Latin-1 character set use character sets that include non-Latin characters. No meaningful conversion is possible between the non-Latin code points and the Latin code points. Arabic, Greek, Hebrew, and Turkish are SBCS languages with non-Latin characters.
- When database sharing takes place, define your files with the CCSID of the primary language being used. Make sure all users have the CCSID of the language that they use defined in their user profile.

See the following for additional information on CDRA:

- Work with CCSIDs
- Use of the SNDNETF command
- Scenario: Multilingual single system
- Scenario: Multilingual network

Use of the Send Network File command: When you use the Send Network File (SNDNETF) command, the data (if sending a member only) is assumed to be in the CCSID of the job that is running the command. Therefore, no conversion takes place. When the data is received, care must be taken to store the member in a file with the same CCSID as the originating file. If the receiver does not know the CCSID of the incoming file member, it can be received into a file with a CCSID of 65535, which indicates that no conversion takes place.

Scenario: Multilingual single system: The following figure shows a multilingual single system with German as the primary language and English and French as secondary languages. All users enter data into the same database file.

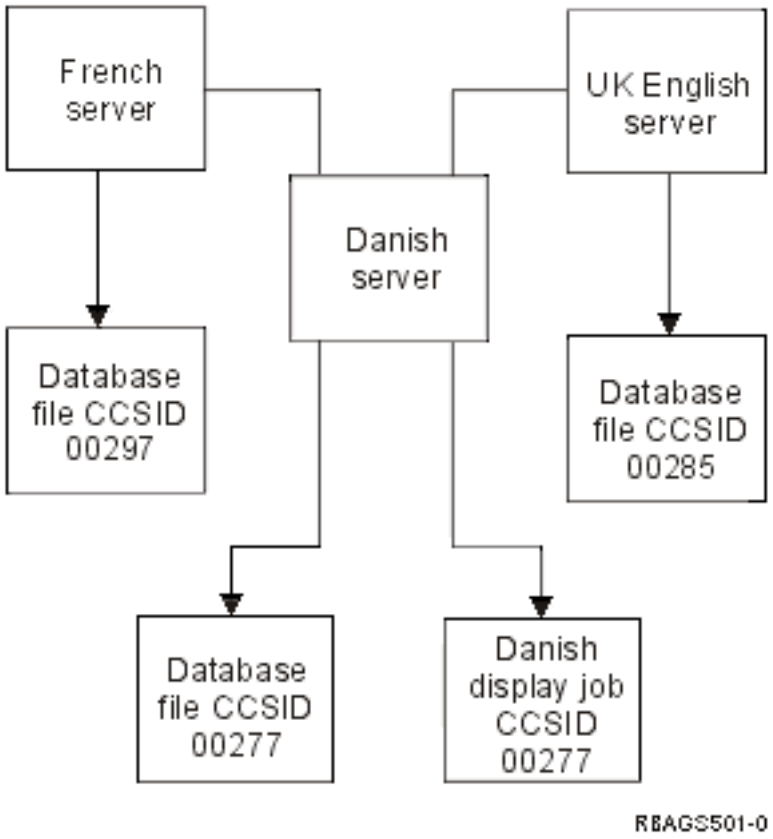


On this multilingual system, all users are entering character data into a single file with CCSID 00273 (German), and character data entered from the English and French display stations is being mapped into the German file.

To preserve correct mapping, fields defined as character fields should be actual character fields. If the fields contain application development values (for example, control characters or fields that are not used as real character fields), the fields either should be specified as hexadecimal fields or assigned a CCSID value of 65535.

Using CCSIDs, characters that cannot be converted between different code pages are replaced with a substitution code. If you are using a user-defined data stream (UDDS) to format and lay out your display (instead of using DDS), you may get substitution codes returned after the system reads and inserts that data in your user-defined data stream. Substitution codes may cause unpredictable results on the display.

Scenario: Multilingual network: The following figure shows an example of a multilingual network with three iSeries servers located in three different countries, each with a different language. In this example, the application on the Danish system is using distributed relational database. All national characters (regardless of the language that the data is stored in) are displayed correctly at the Danish display. When the CCSID of the language is used by the database, the integrity of the database is preserved. The conversion of data between the different code pages is completely automatic and part of the OS/400 database management.



Handle languages with no NLV support: If you need to support a language that does not have a supported national language version, follow these general steps.

1. Study the available national language versions. Find out which national language version most closely resembles your language in character representation.
2. Install the most appropriate national language version as your primary language.
3. Modify the system values to meet your cultural needs. For example, set date and time formats to meet those of the culture that you are supporting. For information on setting system values for cultural needs, see System values for other languages.
4. Configure your workstations and printers to match your primary language. Then, handle discrepancies between support for the installed NLV and your own language.

   Note: The workstation customization functions can support only those capabilities built into your hardware. You cannot support functions through workstation customization that your hardware is unable to support.
5. Use the Create Table (CRTTBL) command to create a sort sequence table based on the existing table that most nearly matches the appropriate sorting sequence for your language.
6. If your language is a DBCS language, create your own characters (UDC) to represent missing characters in the code page associated with the NLV you installed. UDC is an acronym for a user-defined character that is created through the character generator utility (CGU). CGU is an extension of the code page with special user-defined ideographic characters, symbols, or logos.

Programming considerations in global application design

As you develop your global applications, the national language version environment often requires that you pay additional attention to how you prepare and compile your code. The following topics describe some of these requirements, and offer guidelines that you can follow to minimize problems:

- Code globalized applications with high-level languages

- Code globalized applications that use bidirectional data
- Use message catalogs

Code globalized applications with high-level languages

Your major goal must be to have only one general set of running code that is common for all language versions and to make your programs table-driven as much as possible. You should:

- Base validity checks on database accesses and message files rather than on hard-coded literals or tables.
- Base calculations on variable factors retrieved from a file rather than coding them inline.
- Place culture-dependent functions into separate modules of the application and call them when you cannot code them flexibly.

Do not use hard-coded values unless they are fully language and culture independent on comparison, scan, replace, or call operations. In addition, do not use uppercase or lowercase-sensitive values. For example, never hard code Yes and No (Y or N) responses in your program, because these values are different for every language, and should be part of the textual data.

For literals and constants in source code, use characters only from the invariant character set. If input data is checked for validity in the program, make sure that the characters checked belong to the invariant character set; otherwise you may get a situation where the user is requested to enter a character that is not even on his keyboard. For example, the left brace ({) and right brace (}) do not appear on Arabic keyboards. See Invariant character set for a listing of the invariant character set.

Do not use compile-time arrays to hold messages or any other language or culture-sensitive data.

For better performance, when you need to call external NLS-dependent modules, call them by a fixed name as a literal (but based on the library list) rather than by a variable field containing the program name. This allows your application to call the modules of different libraries based on the associated library list.

To allow users to work with an application in the language and habits of their culture, specify the editing values (for example, date, time, and date separators) as dependent on the language and country or region. You can then retrieve them according to the information in the user profile. The parameters are LANGID (language identifier) and CNTRYID (country or region identifier). You need to retrieve the culture-sensitive information only once at program initiation. You can do this by an initial CL program or by the high-level language program and prepare them as:

- Parameters on the call operation
- Parameters on the local data area (LDA)
- Program load tables

Using an initial program allows you to set the user's job attributes to present a consistent application, such as the OS/400 program and other licensed programs.

For additional information about high-level languages, see the following:

- Language compilers CCSID
- Session manager
- ILE C/400(R) considerations
- ILE RPG sort sequence
- ILE COBOL sort sequence
- DB2 and SQL sort sequence
- iSeries Access sort sequence

Language compilers CCSID: Some language compilers expect syntactical operators and the naming convention for the source code to be in CCSID 00037. (Refer to the documentation for the language

compiler you use.) For these compilers, incorrect mapping occurs if the source is compiled with a CCSID other than 00037 or 65535. You must ensure that these compilers receive any variant characters used in language syntax in CCSID 00037.

ILE language compilers

When compiling an ILE C/400, ILE RPG for iSeries, or ILE COBOL for iSeries program, source from database source files is converted to the CCSID of the primary source file.

Compilers for these languages can handle syntactical operators in most CCSIDs. These compilers can also handle naming conventions for the source code in most CCSIDs.

Non-ILE language compilers

When compiling a non-ILE CL, RPG, or COBOL program, source from database source files is converted to the CCSID of the job.

If you do not want your names, constants, or literals converted to the CCSID of the job, you may change your job CCSID to 65535. Your constants, literals and names then remain intact.

Note: REXX/400 procedures and the literal data coded within them are not converted to the job CCSID.

Example 1

The following example shows a sample non-ILE RPG program. This example shows English source on a system in the United States.

```
*  RPG Source   (Source file created using CCSID 00037 but tagged
*                with CCSID 65535)
FFILE1  IF  E                    DISK                          80
C                     READ FILE1
C* Test char
C*
C           FLD1      IFEQ '$'
C                     ...
C* Move char
C*
C                     MOVE FLD1         FLD$
C                      ...
C*
C                     SETON                                   LR
```

Example 2

In Finland, the program in the first example does not compile because the field name FLD$ contains a variant character (the dollar sign). The variant character represents a different code point in a code page other than 00037. This figure shows the same sample non-ILE RPG program as English (U.S.) source on a system in Finland (CCSID 278).

```
*  RPG Source   (Source file created with CCSID 00037, but tagged
*                with 65535)
FFILE1  IF  E                    DISK                          80
C                     READ FILE1
C* Test char
C*
C           FLD1      IFEQ ''
C                     ...
C* Move char
C*
C                     MOVE FLD1         FLD
```

```
C                     ...
C*
C                   SETON                                     LR
```

Example 3

You can correct this error by changing the file CCSID to 00037 and setting the job CCSID to 00278 (for Finland). The following example shows the changed file as seen English source in Finland.

```
*  RPG Source   (Source file created using CCSID 00037 and tagged
*                with CCSID 00037)
FFILE1  IF  E                   DISK                       80
C                   READ FILE1
C* Test char
C*
C           FLD1    IFEQ '$'
C                   ...
C* Move char
C*
C                   MOVE FLD1         FLD$
C                    ...
C*
C                   SETON                                     LR
```

Session manager: For all applications that use a session manager, you must ensure that the output data stream has no X'3F' values in it. OS/400 uses X'3F' values to blank out a screen.

General sort sequence

The sort sequence used by a program may influence the program logic. The following figure shows an example of this.

Using the Latin 1 shared-weight sort sequence, character test 3 is equivalent to character test 4 (not all characters are shown). When using hexadecimal or unique sorting, they are completely different. The following example shows an RPG program using different sort sequences.

```
*  RPG Source   (Program created with Latin 1 sort sequence)
*
C* Test char 3
C*
C           FLD1    IFEQ 'a'
C                   ...
C* Test char 4
C*
C           FLD1    IFEQ 'a'
C           FLD1    OREQ 'A'
C           FLD1    OREQ ''
C           FLD1    OREQ ''
C                    ...
C*
C                   SETON                                     LR
```

If you compile the program with *JOBRUN specified for the SRTSEQ parameter and *JOBRUN specified for the LANGID parameter, the sort sequence table used at run time is not known at compile time.

ILE C/400 and DB2 Query Manager and SQL Development Kit for iSeries licensed programs have additional special considerations.

ILE C considerations: Consider the following when you compile programs with ILE C:

- You can compile a source file in any EBCDIC code page except code page 00290.
- If the CCSID of the primary source file is 65535, code page 00037 is assumed.
- All secondary source files are converted to the CCSID of the primary source file.

Note: While most secondary source files are converted to the CCSID of the primary source file, some conversions are not supported. Contact your IBM service representative if you require support for an unsupported CCSID conversion.

- If the CCSID of the secondary source files is 65535, no conversion takes place.
- Any modules are created in the code page of the primary source file. A module is an OS/400 object that can be a collection of one or more procedures and one or more definitions for external or internal variables. A module is compiled from source code.
- When binding modules of different CCSIDs, no conversion takes place and unpredictable results may occur.
- You can use the trigraph support for the C characters that are not available on all keyboards. Trigraph support generally uses invariant characters to represent variant characters. For example, the left bracket ([) is represented by ??(.

The ILE C run-time library functions that parse strings containing variant characters use the variant character code point value associated with the CCSID of the job.

ILE RPG sort sequence: The ILE RPG for iSeries licensed program provides the possibility for a user to specify a sort sequence table and use it in comparison operations performed with non-numeric data. For each of the supported languages, two tables (a shared-weight and unique-weight) are shipped with the server. With sort sequence support you can create sort sequence tables based on the existing ones.

The control specifications are specifications that provide the ILE RPG for iSeries compiler with information about your program and your server. The sort sequence used in ILE RPG for iSeries programs is controlled by all of the following:

- The control specifications.
- The SRTSEQ (sort sequence table) parameter on the Create RPG Module and the Create Bound RPG Program commands.
- The LANGID (language identifier) parameter on the Create RPG Module and the Create Bound RPG Program commands.

The alternative collating sequence field (ALTSEQ) in the control specifications allows the following values:

blank No alternative collating sequence is used in the RPG program. The normal collating sequence is used in the RPG program. The compile options SRTSEQ and LANGID are ignored.

***NONE**
No alternative collating sequence is used in the RPG program. The normal collating sequence is used in the RPG program. The compile options SRTSEQ and LANGID are ignored.

***SRC** The alternative collating sequence is used in the RPG program, according to the tables entered at the end of the RPG program. The alternative collating sequence table is loaded at compile time, and ordering, sorting, comparing, and match field processing is done according to that table.

The SORTA and LOOKUP operation codes do not use specified alternative collating sequence tables.

The SRTSEQ and LANGID parameters on the Create RPG Module and Create Bound RPG Program commands are ignored.

***EXT** The alternative collating sequence is specified outside of the RPG program. RPG compiler imports an external sort sequence table, based on the SRTSEQ and LANGID parameters on the Create RPG Module and the Create Bound RPG Program commands.

The SORTA and LOOKUP function with the arrays and tables at compile time and processing time take effect only when you specify D in the control specifications.

The sort sequence table to be used by the program can be determined at compile time or when the job is run. If the SRTSEQ parameter of the Create RPG Module and Create Bound RPG Program commands:

- Is set to *HEX, no sort sequence table is used.
- Specifies a table name, then that table is stored with the program object to be used when the job is run. For system-supplied default sort sequence tables for the supported languages, refer to Sort sequence tables.
- Is set to *LANGIDSHR or *LANGIDUNQ, the shared-weight or unique-weight table for the language determined by the LANGID parameter is stored with the program object. For a list of valid language identifiers, refer to Language and country/region identifiers.
- Is set to *JOB, the SRTSEQ parameter of the compile time job is used to determine the sort sequence. The table is stored with the program object.
- Is set to *JOBRUN, the attributes of the job running the compiled program determine the sort sequence to be used. If the SRTSEQ attribute of the job refers to the LANGID, the LANGID stored with the program object is used. If the LANGID stored with the program is also *JOBRUN, the LANGID of the run-time job is used.

Notes:

1. If the table to be stored with the program object at compile time does not exist, a table defining hexadecimal sort sequence and tagged with a CCSID value of 65535 is used.
2. If the sort sequence table and the CCSID of the job running the program differ, the table is converted to the CCSID of the job.

SORTA and LOOKUP operation codes

The implementation of compare operation codes, match field and control field processing with the sort sequence tables is the same for the alternative collating sequence and for the sort sequence support. Compare operation codes are ANDxx, COMP, CABxx, CASxx, DOUxx, DOWxx, IFxx, ORxx, and WHxx. Additional functions provided with the SORTA and LOOKUP operation codes follow:

SORTA
The data in the array is sorted according to the sort sequence table.

LOOKUP
To provide proper table searching, the sort sequence table is used with the search arguments in the arrays and tables.

The search argument and either the table or array element are compared using the sort sequence table.

The array and table data are checked using the sort sequence table, whenever ascending or descending sequence is specified. If the SRTSEQ and LANGID parameter values resolve to retrieve the sort sequence table again at run-time, then the array and table elements are loaded without a sequence check at the compile time. The sequence checks are performed at run time, according to the sort sequence table.

ILE COBOL sort sequence: The ILE COBOL for iSeries licensed program uses the sort sequence support in the following ways:

- Create COBOL Module command
- Create Bound COBOL Program command
- PROCESS clause
- ALPHABET clause

The ILE COBOL for iSeries licensed program uses sort sequence tables that are system-supplied or user-supplied.

Create COBOL module and create bound COBOL program commands

These CL commands have two compiler options relating to sort sequence support: the SRTSEQ parameter and LANGID parameter. The SRTSEQ parameter allows the user to specify any of the system-supplied or user-supplied sort sequence tables residing in a specified library. You can specify whether the sort sequence table should be taken at compile time or run time. Also, you can choose between the shared-weight and unique-weight tables.

With the LANGID parameter, you can specify one of the system-defined language identifiers, or leave that parameter to be defined at the run time.

The meanings of the SRTSEQ and LANGID parameters on the Create COBOL Module and Create Bound COBOL Program commands are the same as on the Create RPG Module and Create Bound RPG Program commands as described in “ILE RPG sort sequence” on page 46.

PROCESS statement

Sort sequence support options can be supplied in the PROCESS statement. The syntax for that command is like that for the Create COBOL Module and Create Bound COBOL program commands. The only exception to this is that the values for the parameters in the PROCESS statement are entered without an asterisk (*) for the predefined values. Any options specified in the PROCESS statement override the corresponding options on the Create COBOL Module and Create COBOL program commands.

ALPHABET clause

The alphabet-name in the ALPHABET clause of the SPECIAL-NAMES paragraph may use the NLSSORT option. Use the SRTSEQ and LANGID parameters of the compiler for alternative collating sequence options. Otherwise, it means the same as the NATIVE option.

The following COBOL lines are affected by the NLSSORT option:

- PROGRAM COLLATING SEQUENCE phrase of OBJECT-COMPUTER paragraph

  When evaluating the result of nonnumeric comparisons, the alphabet name has to be referenced in this phrase to enable the program to use the specified sort sequence options. This option also applies to the nonnumeric sort or merge operation. Otherwise, the hexadecimal collating sequence is used.
- ALPHABET CLAUSE in the SPECIAL-NAMES paragraph

  This clause should specify the NLSSORT option.
- COLLATING SEQUENCE in the MERGE (or SORT) statement

  This phrase is used to specify the collating sequence to be used for nonnumeric comparisons for the KEY data name in the MERGE or SORT operation. If omitted, the PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph defines the collating sequence to be used. If neither is specified, hexadecimal collating sequence is used.
- Nonnumeric relation names and condition names

  The selected sort sequence table affects the result of certain statements, using nonnumeric relation names and condition names: EVALUATE, IF, PERFORM...UNTIL, SEARCH and START. The truth values of the nonnumeric comparisons depend on the corresponding weights of the characters in the selected sort sequence table. For example, if you specify unique-weight table (LANGIDUNQ) for French (Latin 1), the following statement is true for the single value of the variable ITEM-1,e.

  ```
  IF ITEM-1 = "e"
  ```

  If you specify a shared-weight table (LANGIDSHR) for French (Latin 1), the same statement is true for several values of the variable ITEM-1. All have the same shared weight of 77:

  ```
  lowercase e (e), uppercase e (E),
  lowercase e acute (), uppercase e acute (),
  lowercase e grave (), uppercase e grave (),
  lowercase e caret (), uppercase e caret (),
  lowercase e umlaut (), uppercase e umlaut ()
  ```

DB2 and SQL sort sequence:
For interactive SQL, the SRTSEQ and LANGID parameters can be specified on the STRSQL command. Later these parameters can be changed using the session services for interactive displays.

Sort sequence tables are used for all string comparisons. String comparisons are performed in the following SQL statements:

- ORDER BY clause
- WHERE clause
- GROUP clause
- HAVING clause
- UNION and UNION ALL clauses
- DISTINCT clause
- BETWEEN predicate
- IN predicate
- LIKE predicate
- MIN and MAX scalar functions
- MIN and MAX column functions

In addition, any indexes or views that are created using the CREATE INDEX or the CREATE VIEW statements are created with the specified sort sequence table.

DB2 Query Manager and SQL Development Kit for iSeries

DB2 Query Manager and SQL Development Kit for iSeries does not assume a particular CCSID when precompiling source. Any variant characters in the language syntax (such as the

¬

not symbol) are assumed to be encoded in the CCSID of the source file.

For example, if the source file has a CCSID of 00037, the

¬

not symbol is correctly interpreted to be at code point X'5F'. If the source file has a CCSID of 00500, however, the

¬

not symbol is correctly interpreted to be at code point X'BA'.

A literal is stored in the CCSID of the source file.

Because DB2 Query Manager and SQL Development Kit for iSeries calls the appropriate language compiler to create an SQL program, the general guidelines for high-level languages must also be taken into account.

iSeries Access sort sequence: You can specify the sort sequence in the iSeries Access functions, remote SQL and transfer function. When performing queries on the server databases and SQL tables, you can specify the system-supplied or user-supplied sort sequence tables.

Remote SQL support

You can specify the way the selected data has to be sorted when performing the query. For that purpose, sort fields have to be specified in the ORDER BY clause. The following clauses also use the specified sort sequence:

- WHERE clause
- GROUP BY clause
- HAVING clause
- JOIN BY clause
- UNION clause
- DISTINCT clause
- IN predicate
- LIKE predicate
- BETWEEN predicate
- RANGE predicate
- MAX function
- MIN function

The actual sort sequence table is retrieved from the job attributes of the user. The SRTSEQ and LANGID parameters can be affected through changing the user profile or changing the job attributes.

Transfer function support

When transferring data from the iSeries server to the workstation, you can specify the sort sequence to be applied on selected data. The sort sequence table is also used in the following string comparison operations:

- WHERE clause
- GROUP BY clause
- HAVING clause
- JOIN BY clause
- IN predicate
- LIKE predicate
- BETWEEN predicate
- MAX function
- MIN function

You can specify in the OPTION statement the following parameters related to sort sequence:

- SRTSEQ (sort sequence table)
  - *JOB
  - *HEX
  - *LANGIDSHR
  - *LANGIDUNQ
  - *LIBL/sort-seq-table-name
  - *CURLIB/sort-seq-table-name
  - library-name/sort-seq-table-name
- LANGID (language identifier)
  - *JOB
  - language-identifier

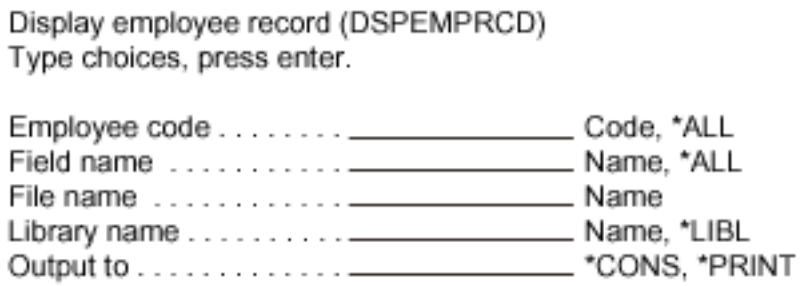
You can choose the appropriate sort sequence through options on iSeries Access displays.

Code globalized applications that use bidirectional data

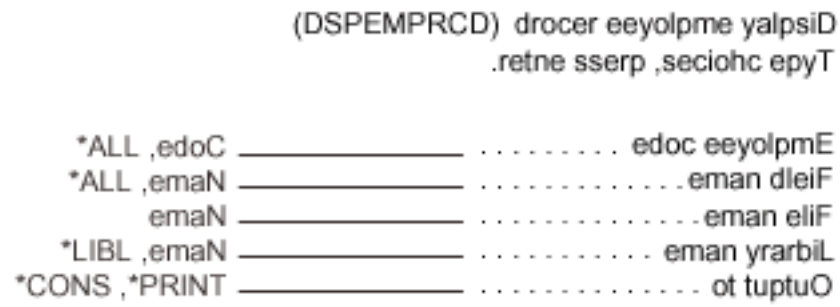
When you are developing NLV-enabled applications, consider the following restrictions:

- Bidirectional language display layout
  The presentation of data should have a right-to-left orientation. Literals should appear on the right side of the fields that they describe. The following examples illustrate a U.S. English display with a left-to-right orientation and the same display in a right-to-left orientation.

  Left-to-right layout of a U.S. English display



  Right-to-left layout of a U.S. English display



- Long fields in bidirectional languages
  Avoid defining input fields that span more than one line. When the field is displayed or printed as one entity, the result for bidirectional languages would not be what the user intended.
- Variable positioning in bidirectional languages
  Your application must allow for variables to be in any order. For example,consider the following message in English:

  ```
  File &1 in library &2 not found
  ```

  When translated to another language, the message might look like the following:

  ```
  dnuof ton &2 yrarbil ni &1 eliF
  ```

  In this case, variable 2 is positioned before variable 1.
- CHECK(RL) and CHECK(RB) keywords with bidirectional languages
  These options are valid only for display stations capable of right-to-left movement, and have the following restrictions:
  - Option indicators are not valid with cursor control codes.
  - CHECK(RZ) and CHECK(RB) are not valid with these keywords.
  - A field that spans more than one line gives a warning message.
  - The check digit for modulus checking is the farthest-right byte in the field.

  - CHECK(RL) applies to character fields only.
- Online information for bidirectional languages
  The special bidirectional tags have a restriction. When combining online help information from several panel groups that do not have the same value for the BIDI tag, the end user must use the hot key sequence to read the opposite orientation online help information.
- CCSIDs for bidirectional languages
  As bidirectional languages have special character sets that are unique, no exchange of data into other languages is feasible. You may need to use data mapping between EBCDIC and ASCII data streams, however. For example, you need data mapping between EBCDIC and ASCII data streams if you are using Distributed Relational Database Architecture(R) (DRDA(R)).

  When exchanging data in a language that uses Latin characters and when special characters that are not part of the invariant character set are needed, use CCSID 00424 for Hebrew and CCSID 00420 for Arabic for data mapping to take place. For a list of supported CCSIDs, see CCSIDs.

See Work with bidirectional data for more information about bidirectional data. This topic also includes a checklist for bidirectional data that provides guidelines to follow when you create applications with bidirectional support.

Use message catalogs

OS/400 can use message catalogs to store messages. Messages in a message catalog are grouped as sets. Each message has a unique number within a set. You can create a message catalog as a stream file, a source file member, or a user space object type from one or more source files.

Because you can store message catalogs as stream files, you can use directories to isolate messages for specific products or national language versions.

Create or update a message catalog with the GENCAT and MRGMSGCLG commands

You can use both the Generate Message Catalog (GENCAT) command or the Merge Message Catalog (MRGMSGCLG) command to create or update a message catalog. Once a message catalog exists, continued use of these commands updates a catalog by comparing the original messages to the messages in the source. New message text replaces specific messages without changing the other messages within the set. With these commands, you can add or delete messages from an existing set of messages. You can also delete sets of messages from an existing message catalog.

For more information

See the following topics for more information about message catalogs:

- Source for message catalogs
- Open, extract, and close message catalogs

Source for message catalogs: The source for a message catalog is either a source physical file, a stream file, or multiple files. The source contains fields to define set numbers, message numbers, message text, or to specify sets to delete. The following topics provide additional information and examples relating to message catalogs.

Message catalog source format

A message catalog contains five fields of message text source lines. A single blank character separates each of the five fields. Any other blank characters are considered as part of the subsequent field data. See Special characters and escape sequences (see page 54) for additional information.

Note: Enter the key fields exactly as listed below, using the dollar sign ($) and lowercase characters. Definitions for maximum and minimum values are stored in QSYSINC/QSYS/LIMITS.

- **$ comment**
  A line that begins with $ that is followed by one or more blank characters is treated as a comment line. A omment line should be placed directly beneath the message to which it refers. Place comments for an entire set directly below the $set directive in the source file.
- **$quote** *C*
  This line specifies an optional quote character *C* that is used to surround message text. This character enables trailing spaces or null (empty) messages to be visible in a message source line. By default, or if an empgy $quote directive is supplied, no quoting of message text is recognized.
- **$set n comment**
  This line specifies the set identifier of the messages to follow until the next $set or end-of-file appears. The *N* denotes the set identifier that is defined by a number between 1 and NL_SETMAX. Place set identifiers in ascending order within a single source file. They do not need to be contiguous. A character string that follows a set identifier is treated as a comment and ignored.
- **$delsetncomment**
  This line deletes message set *n* from an existing message catalog. The *n* specifies the set number. Data that follows the set number is treated as a comment. The $set and $delset identifiers can both be in the message catalog source or the field tags.
- **m message text**
  The *m* specifies the message identifier that is defined by a number between 1 and NL_SETMAX. The message text is stored in the message catalog with message identifier *m* with the set identifier that is specified in the last $set directive. If the message text is empty and a blank character field separator is present, it stores an empty string in the message catalog. Existing messages get deleted from the catalog if the message line does not have a field separator or MESSAGE TEXT and a NEWLINE or carriage return follows the message line. Message identifiers must be in ascending order, noncontiguous, and within a single set. The length of the MESSAGE TEXT must be in the range of 0 to NL_TEXTMAX.

  Note: Empty lines in a message text source file will be ignored.

Messages programming format

MESSAGES should follow these recommendations:

- The last line of all messages should end with \n.
- The second and remaining lines of a message should begin with \t, indicating a tab.
- All lines of messages that continue to the next line should end with \n\, indicating that the message continues to the next line.
- The quotation mark at the end or beginning of a line should be omitted. The quotation mark delineates the beginning and end of a complete message.

Using multiple source files

You can specify multiple source files for the source file parameter. The messages that are contained in all of the files must follow the same rules for sets and messages as defined in a single source file. For example, the first source file contains messages in sets 1 through 3. The next source file must begin with set 3 and have a message number greater than the last message number in the first source file. If not, it must contain sets that begin with a number higher than the highest number (set 3) in the previous source file.

Replacing messages

Messages in an existing message catalog can be replaced by specifying a source file that contains the same set number and message number as the message text you want to change. All other messages in the source file remain the same. To update a value for the $QUOTE in a catalog, use the same $QUOTE character in subsequent source files.

Example source for a message catalog

The following shows a sample format for the source that is used to create a message catalog. A quotation mark delineates each message. The message text that is stored in the message catalog has had the extraneous blank characters removed. This example describes three sets of messages. Set 2 is deleted while sets 1 and 3 remain stored in the message catalog.

```
$ Messages for my new product
$quote "

$set 1

1 "Error occurred.\n"
$ The next message is continued on the next line.
2 "This is a very long message \n\
\t that requires another line to display. \n"
3 "Specify a value greater than %d.\n"
4 "File %c cannot be used at this time.\n"

$set 2
1 "Error %d occurred. \n"
2 "Flag not set.\n"
3 "Number of arguments must be %d.\n"

$set 4
1 "Before using this command, you must \
set the correct values in the %c box.\n"
2 "You have not properly NLS enabled this function.\n"
10 "Messages should end with a %c.\n"

$delset 2
```

Note: Message 2 in set 1 will be displayed in two lines. Message 1 in set 4 will display as a one line message.

The following is an example for using the MRGMSGCLG command to create a message catalog.

```
MRGMSGCLG CLGFILE('/MYPRODUCT/MESSAGES?US')
     SRCFILE('QSYS.LIB/MYLIB.LIB/MYSOURCE.FILE/US.MBR')
     CLGCCSID(*SRCCCSID) SRCCCSID(*SRCFILE)
     TEXT('Message catalog for USA')
```

This example creates a message catalog into the stream file US in directory /MYPRODUCT/MESSAGES using the source from MYLIB library in file MYSOURCE and member US. The CCSID of the data in the message catalog is the same as the CCSID tag of the source file.

Special characters and escape sequences

Text strings can contain special characters and escape sequences as defined in the following table.

Description of special characters	Sequence
\	\\
backspace	\b
carriage return	\r
form feed	\f
horizontal tab	\t
NEWLINE	\n

Description of special characters	Sequence
octal bit pattern	\ddd **Note:** The escape sequence \ddd consists of a backslash followed by up to three octal digits that specify the value of the desired character. If the character following the backslash is not an octal digit, the backslash and data following are included as part of the text.

Open, extract, and close message catalogs: Once you have created a message catalog, you can use the following C functions:

CATOPEN()
Opens a message catalog

CATGETS()
Extracts a message from a message catalog, given a set identifier and a message identifier

CATCLOSE()
Closes the message catalog

The C function CATOPEN opens the message catalog. If no slash (/) characters are found in the name, the NLSPATH environment variable and the LC_MESSAGES category are used to find the specified message catalog. If the name contains one or more slash (/) characters, the name is interpreted as a path name of the catalog to open.

A default path is used if there is no NLSPATH environment variable or a message catalog cannot be found in the NLSPATH path specified. If the value of oflag is NO_CAT_LOCALE the environment variable setting of LC_MESSAGES may affect the default path. If the value of oflag is zero the LANG environment variable may affect it also.

For more information about C functions and message catalogs, see the ILE C/C++ Language Reference

PDF.

Deliver globalized applications

As you prepare to deliver your global application, you should consider how globalization issues might affect the ways that your customers will install and use your application. The following topics briefly discuss these issues.

Hardware support for multilingual systems

Hardware, in this context, means the physical keyboards, displays, printers, and controllers that make up an iSeries server. The extent to which this hardware supports national languages may impose limitations on the degree of support that you can provide with an application. You must refer to the reference manuals for non-IBM hardware to determine what limitations, if any, are imposed by that hardware.

Character data translation

Translating is changing the meaning of character data from a set of concepts, ideas, and statements in one human language to a culturally similar meaning in another human language. You can follow some basic rules to ensure translation goes smoothly. A subset of these rules is provided in the “User interfaces” on page 14 topic.

Delivering your globalized application to customers

Delivering your application to customers includes the processes of packaging, servicing, supporting, and educating users about your application. You must consider various tasks when following these processes in different countries and cultures throughout the world. See “Packaging and installation process” on page 6 for more information on the processes associated with the delivery of your application to customers.

Printed in U.S.A.