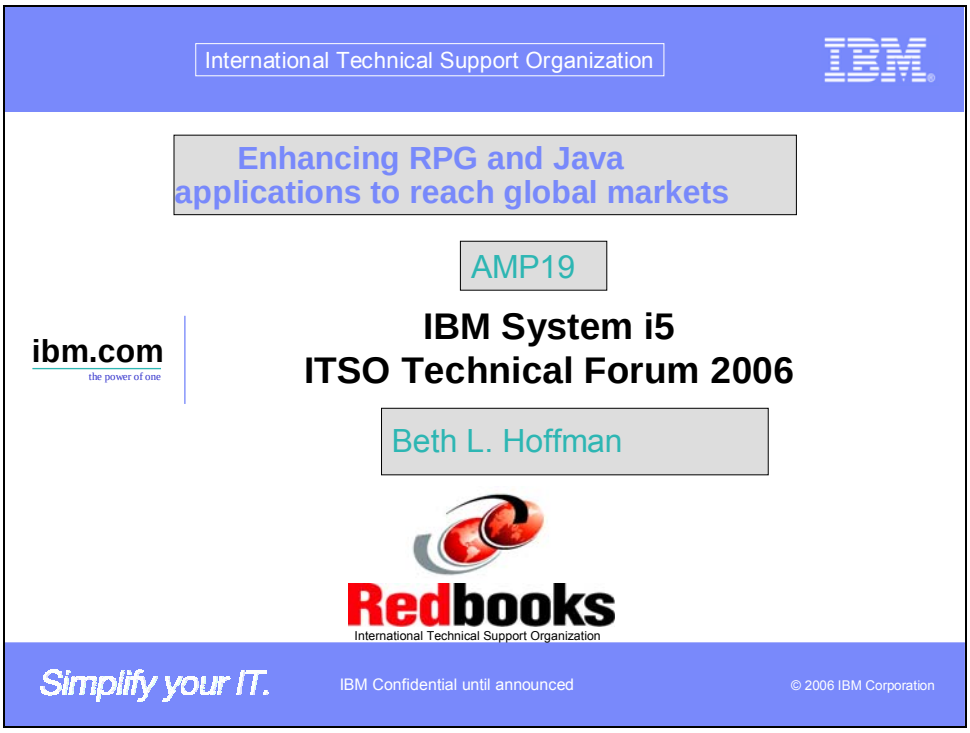
Table of contents

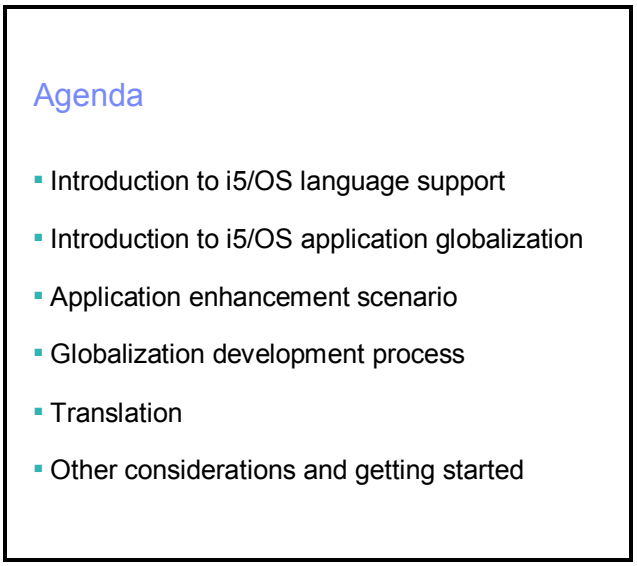
About the author

Beth L. Hoffman has worked as a software engineer for IBM in Rochester, Minnesota, for more than 15 years. For much of that time, she led the development of key middleware products on the IBM OS/400 and IBM i5/OS operating environment. Today, she is a technical consultant working with solution providers who are enhancing their applications with new technologies. You can e-mail Beth at bethvh@us.ibm.com.



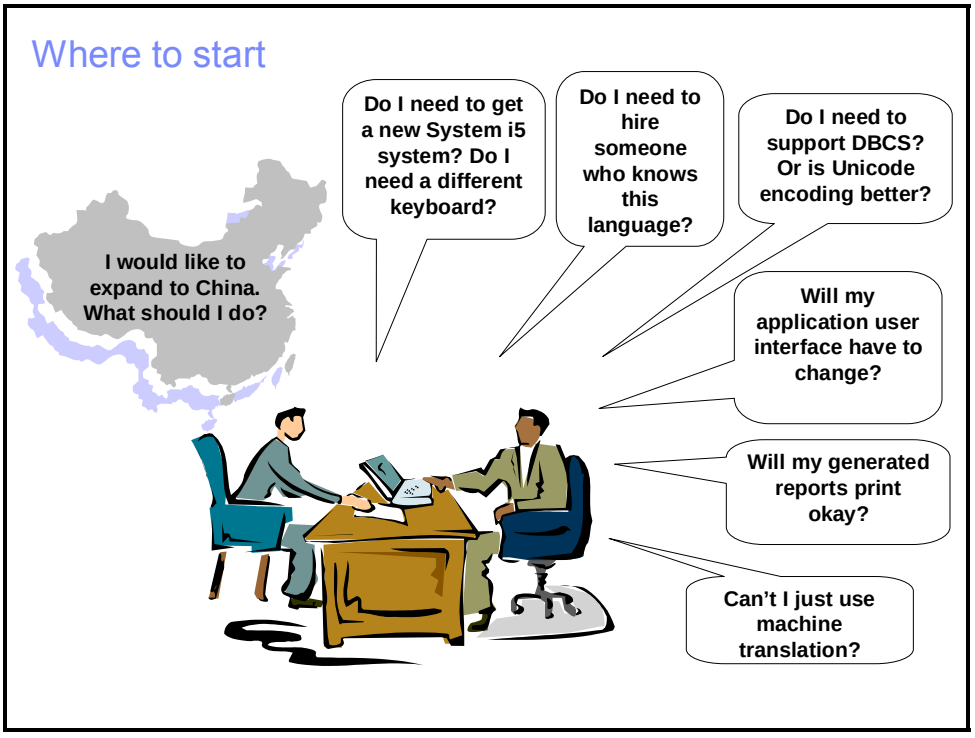
Introduction

Most software in the world today only supports one language. In recent years, many countries have seen a sharp increase in the use of technology. These emerging markets offer a great opportunity for broadening software usage and increasing software sales. Software companies that want to seize these opportunities must prepare their solutions to support other countries and languages. This presentation provides you with a solid introduction to globalization, including such concepts and terminology as the Unicode™ Standard and double-byte character set (DBCS). This course will examine the process for making software applications global from the planning stages through translation and testing. Tips and techniques will provide important insight for user-interface design, data handling, and testing. Additionally, you will see an analysis of code examples for RPG and Java™. By the end of this presentation, you will have a better understanding of what is necessary to make your IBM® System i™ application support multiple languages.



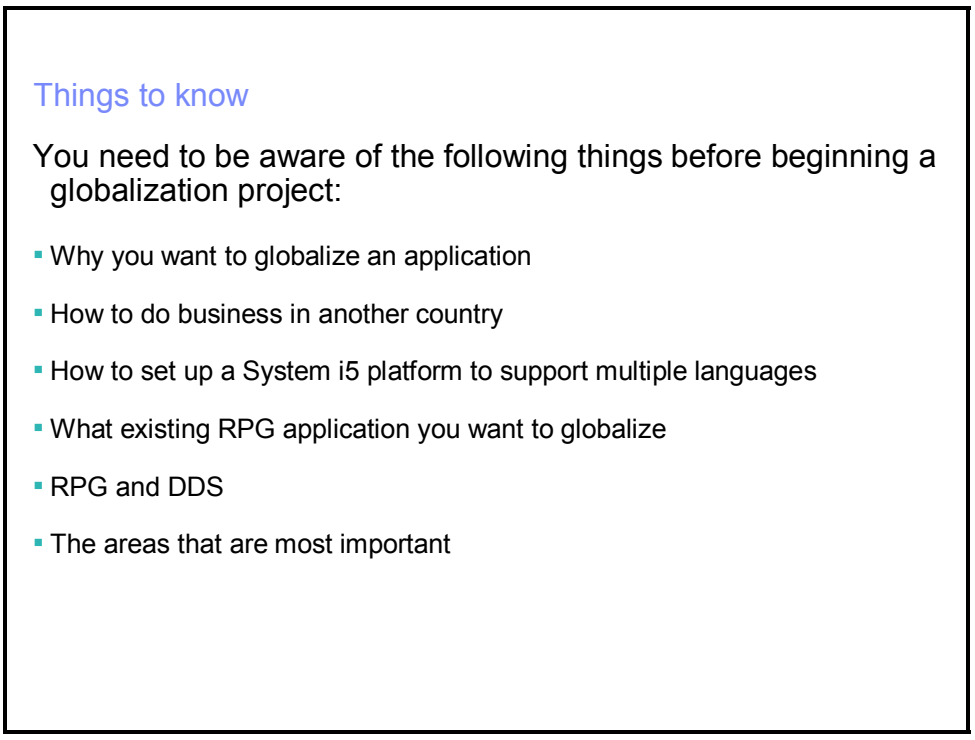
Agenda

This presentation will take approximately 60 minutes, and will flow as shown here. First, the course will introduce you to the concepts of language support and globalization on your System i platform. Next, you will review an application enhancement scenario. Then, there will be a discussion of the globalization development process, including translation tasks and issues. Finally, you will read about important considerations when getting started down the path to globalization. Recommendations, tips, and examples appear throughout many sections. Be sure to browse the information in the appendix, which includes additional details, terminology definitions, and pointers to important resources.



Where to start

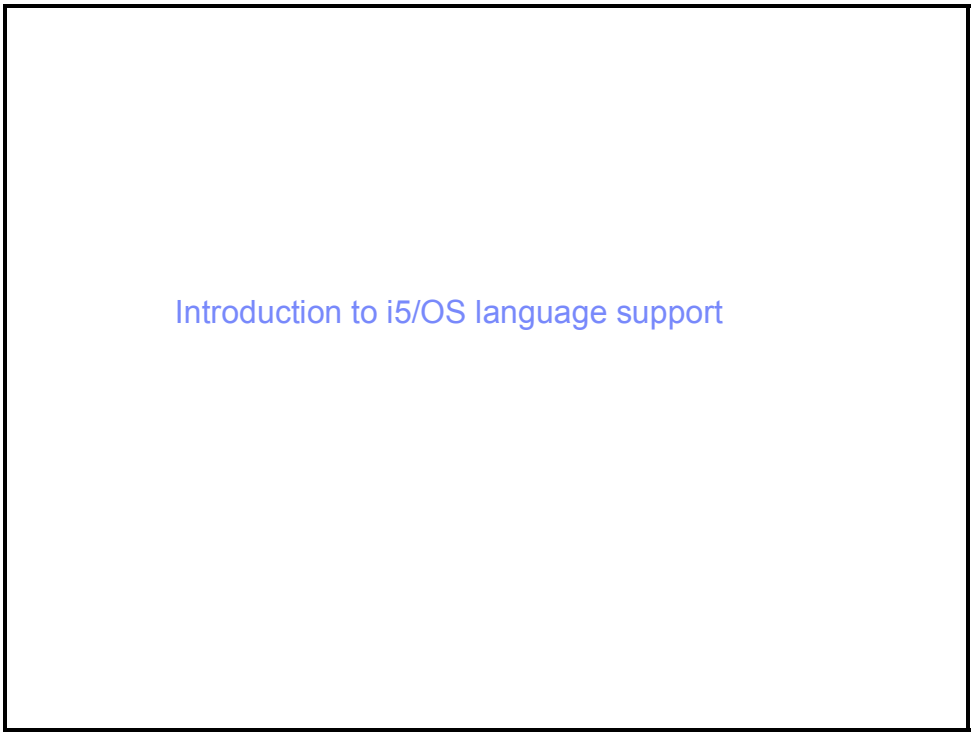
When application providers want to expand the use of their applications into other countries, they often do not know where to start. You might be in this same situation. This presentation will help answer many of the questions that arise, and it will provide you with a good foundation for starting your own application globalization project.



Things to know

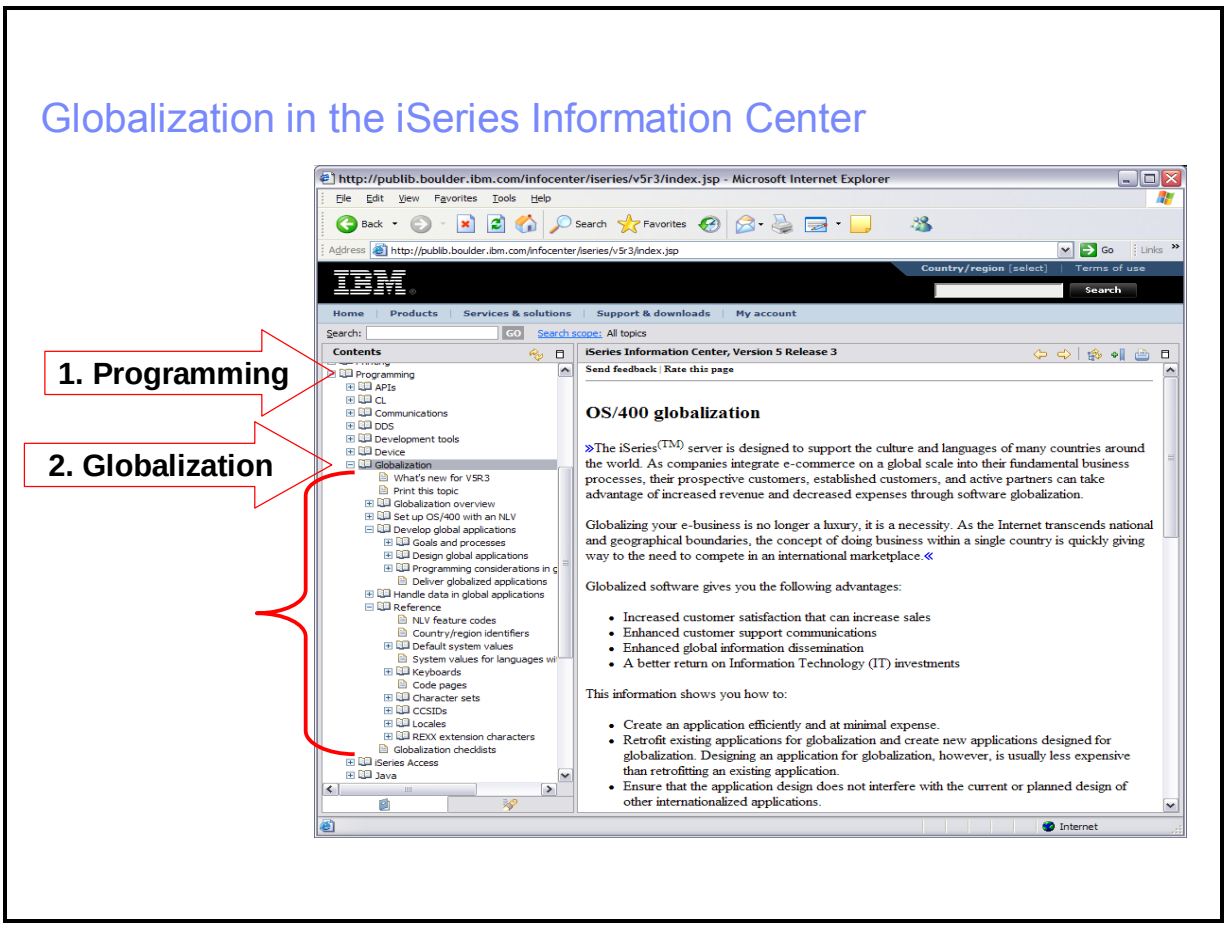
Globalization is a large topic. There are many aspects to discuss. In order to narrow the scope for this presentation, there are some assumptions about what you already know and what you are willing to investigate after the presentation. To benefit the most from this course, you need to know the answers to all the questions shown on this chart.

The main scope of this presentation will be on helping you understand how to enhance existing RPG applications.



Introduction to i5/OS language support

This section describes the base language support in the IBM i5/OS® environment.



Globalization in the iSeries Information Center

The IBM iSeries™ Information Center is the primary place to find information about globalization support in the i5/OS environment. To access the information, click the **Programming** section, and then click **Globalization**. There are many sections that you will want to browse, including the sections titled **Develop global applications**, **Handle data in a global environment**, and **References**.

Other sections in the Information Center provide additional globalization information. For example, the RPG language provides Unicode support in many of its built-in functions. The iSeries Programming Guide documents this information. To find this information, navigate as follows: **Programming->Languages->RPG->ILE->Programming Guide**.

The System i platform delivers global systems

The System i platform is a great system for running global applications. This chart offers an overview of how the system supports multiple languages.

The System i platform supports many languages, and it allows you to use the system in the language of your choice. The main system hardware unit is the same regardless of what languages you use on it. The system includes language feature code, which controls the language you will see when you log on and use your System i platform. When you order a system, you must specify the language feature code for the primary language the system will use.

Generally, i5/OS software uses a common set of program code, regardless of the language you use, but it also includes a set of textual data that is language specific.

The i5/OS operating system provides a national language version (NLV) that contains a predefined set of language-dependent values, such as date format, time format, and sort sequence. You can use more than one language on one system by installing one or more secondary languages, as long as there is adequate storage space. This enables users on the system to select the language each of them wants to use, and it allows the same application to run in different languages. Finally, you can set up separate partitions to run various languages.

To use a DBCS language in releases prior to i5/OS Version 5 Release 3, you must install the operating system and specify the DBCS language as the primary language.

Primary language

- The i5/OS operating system initializes all language-dependent or culture-dependent system values from the primary language.
- Other system objects and functions assume attributes based on the primary language.

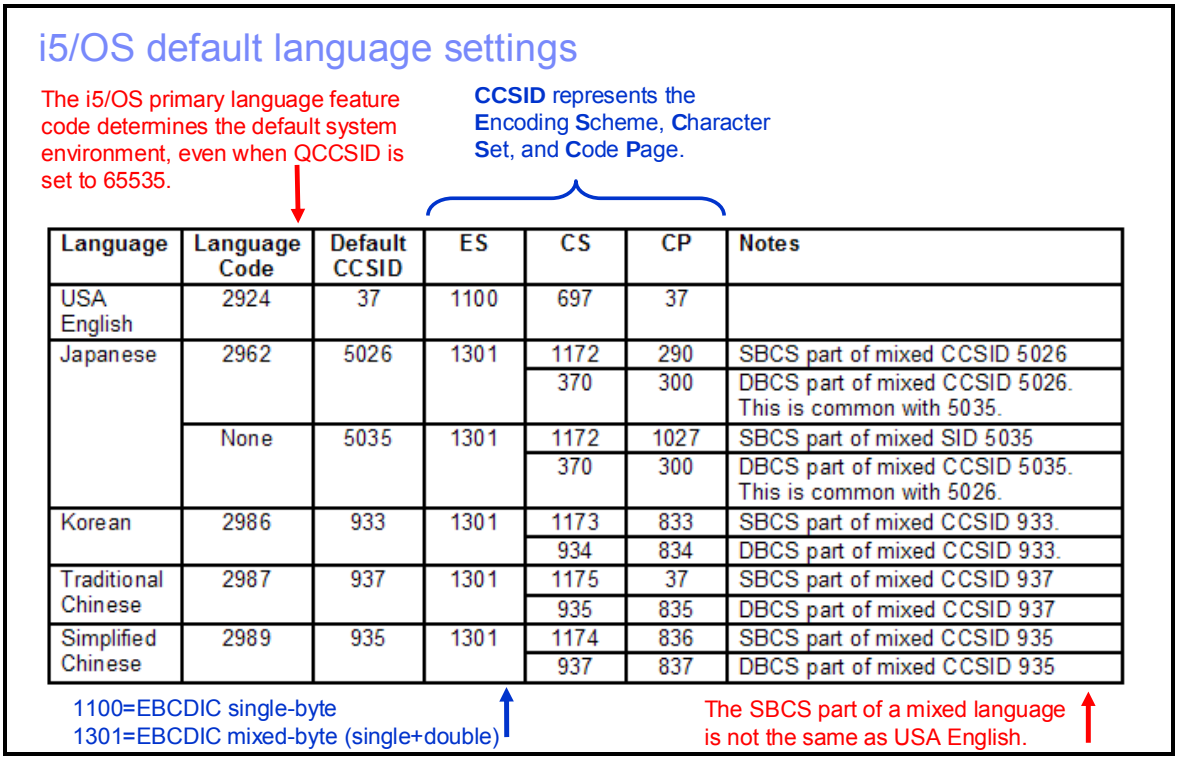
Primary language

The i5/OS operating system derives all language-related system settings from the primary system language. If a user wants to run an environment in a secondary language, several options control which language to use.

The primary language is the language from which all language-dependent or culture-dependent system values are initialized. Other system objects and functions assume attributes based on this language. For example, messages appearing in the history log always appear in the primary language.

In order to use a secondary language for displaying i5/OS functions, choose one of the following options:

- Place the desired language at the top of the library list. For example, to present the French version of textual data, put French information at the top of the library list with the i5/OS command:
  ```
  CHGSYSLIBL LIB(QSYS2928) OPTION(*ADD)
  ```
- Create a subsystem for the secondary language. Define the subsystem in the library list entry with the NLV library for that language. All jobs running in the subsystem use textual data from that language, and all jobs submitted as batch jobs have the NLV library as the first library on the system part of the library list.
- Change the library list for your job so the NLV library for that language is first.



Language	Language Code	Default CCSID	ES	CS	CP	Notes
USA English	2924	37	1100	697	37	
Japanese	2962	5026	1301	1172	290	SBCS part of mixed CCSID 5026
				370	300	DBCS part of mixed CCSID 5026. This is common with 5035.
	None	5035	1301	1172	1027	SBCS part of mixed SID 5035
				370	300	DBCS part of mixed CCSID 5035. This is common with 5026.
Korean	2986	933	1301	1173	833	SBCS part of mixed CCSID 933.
				934	834	DBCS part of mixed CCSID 933.
Traditional Chinese	2987	937	1301	1175	37	SBCS part of mixed CCSID 937
				935	835	DBCS part of mixed CCSID 937
Simplified Chinese	2989	935	1301	1174	836	SBCS part of mixed CCSID 935
				937	837	DBCS part of mixed CCSID 935

i5/OS default language settings

This table shows the default language settings on the i5/OS platform for a few languages. Along the left, you can see English as well as the four DBCS languages. The primary language feature code dictates the default system environment, even when the coded character set identifier (CCSID) system value is set to the default of 65535. Each language feature code has an associated default CCSID. A CCSID is a combination of the encoding scheme, character set, and code page shown in the next three columns.

There are two major CCSIDs for the Japanese language. The majority of Japanese IBM customers use CCSID 5026 because it is the default. CCSID 5035 became another major Japanese CCSID because of its compatibility with the single-byte character set (SBCS) that the English environment uses. The DBCS part of both CCSIDs is the same. Using CCSID 5035 can ease the effort of enhancing SBCS English applications to support Japanese users.

It is important to remember that the default CCSID is 65535, which means the system does no conversion. This allows for compatibility with the system. Most businesses that use only one language with their systems leave the default CCSID as 65535. Companies that run more than one language on their systems might find a need to change the default to something else, such as CCSID 500.

The encoding scheme is a value that specifies whether the data is EBCDIC single-byte, EBCDIC double-byte, or both. (EBCDIC is the native data encoding method for the System i platform.)

It is important to tag all data used by an application with its appropriate CCSID so that the programming language and database environments can handle the data appropriately.

Language-related system values and default values

Language / System Value	DBCA Upper/Lower English	Japanese	Korean	Traditional Chinese	Simplified Chinese
	2984	2962	2986	2987	2989
QCHRID	01175 00037	01172 01027	01173 00833	01175 00037	01174 00836
QDECFMT	Blank	Blank	Blank	Blank	Blank
QKBDTYPE	TAB	JKB	KOB	TAB	RCB
QCURSYM	Dollar ($)	Yen Sign	WON Sign	Dollar ($)	Dollar ($)
QDATSEP	Slash (/)	Slash (/)	Period (.)	Slash (/)	Period (.)
QDATFMT	MDY	YMD	YMD	YMD	YMD
QTIMSEP	Colon (:)	Colon (:)	Colon (:)	Colon (:)	Colon (:)
QCCSID	N/A	05035	00933	00937	00935
QCNTRYID	US	JP	KR	TW	CN
QLANGID	ENU	JPN	KOR	CHAT	CHS

Language-related system values and default values

This table shows the default system values, which are set based on the primary language feature code. Here are examples for the same five languages shown in the previous chart.

Introduction to i5/OS application globalization

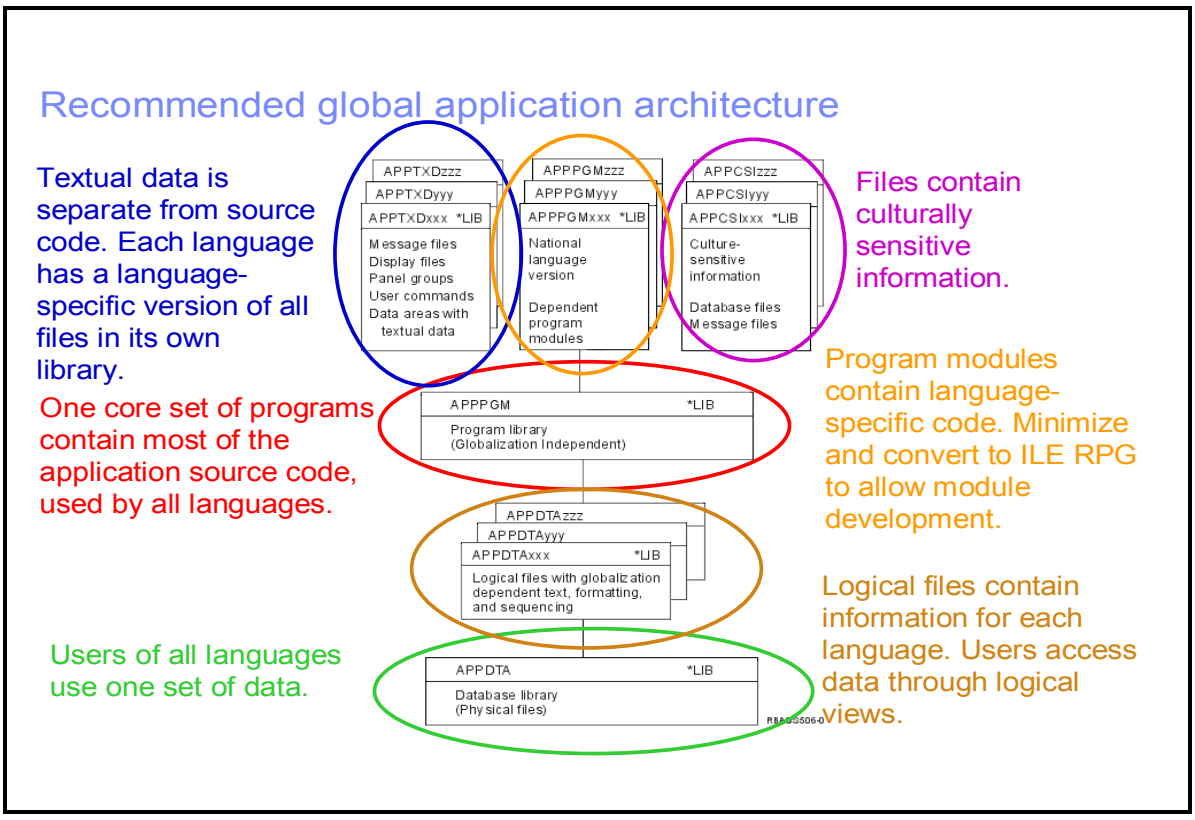
The following section introduces the concept of i5/OS application globalization.

Overall goals for a global application

The primary reason to make an application global is to be able to sell copies of your application to buyers in other countries, regions, or cultures in order to increase your company's revenue. However, it is wise to look at the main goals globalizing an application from different points of view.

Global applications allow each user to work with that application and its related data in their language of choice. The user can expect the application format to honor cultural preferences in its user interfaces. You can see in the chart that four users are working with the same application on one System i machine, each being productive with their preferred environment. This is a truly global application.

The goal for application providers is to deliver this capability with a minimum amount of work and in the shortest amount of time. This means having one set of core programming code and minimizing the number of changes to it. In this presentation, you will see a number of recommendations and tips that will help you create a better globalization project plan and implement the project with the least risk.



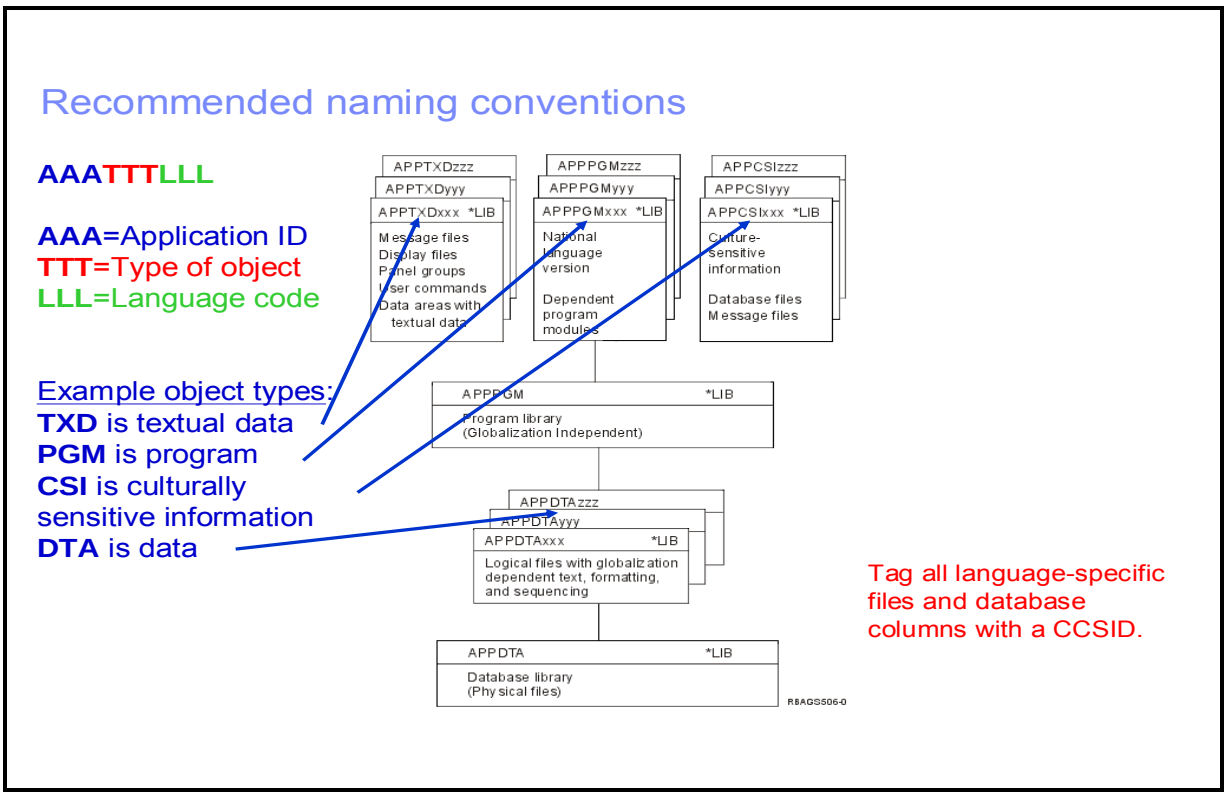
Recommended global application architecture

This chart illustrates the general recommendations for global application architecture.
The goal is to have your core application logic independent of any language. You do this by removing all language-specific elements from the core application logic, such as textual strings, messages, and the description of the user interface. This allows for easier translation of textual information. Good software design requires you to keep the programming code implementing the user interface (UI) separate from the code implementing the underlying functionality. The description of the UI is also separate from the code implementing it. On the System i architecture, this means keeping text strings out of the RPG code and, instead, placing them in the data description specification (DDS) files and sometimes into message files (*MSGF).

You can break down language-specific code into separate code modules if needed. An example of this is special code to calculate taxes for a certain country.

Culturally sensitive information (CSI) refers to any special information that is configurable and language-specific. Storing this outside the program code is important.

You can also push functions out of the code and into the database by using built-in database features and logical files. An example of this is removing sorting processes from the program logic and defining a sort sequence in a logical file to handle it.



Recommended naming conventions

This chart shows some recommended naming conventions as an example. It is beneficial for names of libraries and files to indicate what each contains. With large applications that contain many files, using consistent, useful naming conventions is important, especially if several programmers are working on the code.

The names for the following items need to use invariant characters that are available in all language environments:

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

Supply each language with its own library. Designate the language in the library name. Be sure that each file name in the library includes the language as well. Tag all language-specific files and database columns with the language CCSID. Library lists can help you quickly find the correct files in the right library.

Enhancing an existing RPG application

Four approaches for globalization:

1. **Create a new application using modern technologies.**
2. Create a new version of the application for each language.
3. Enhance the current application.
4. Create a new modern version of the existing application.

Enhancing an existing RPG application

This chart shows some general approaches for making your application global.

Approach 1: The first approach is to create a new application that uses modern technologies. This allows you to take advantage of all the built-in globalization functions in a modern language, such as Java. Rewriting an application, or creating a new one to replace an existing application, can be a huge investment; therefore, this approach is generally not feasible.

Enhancing an existing RPG application (continued)

Four approaches for globalization:

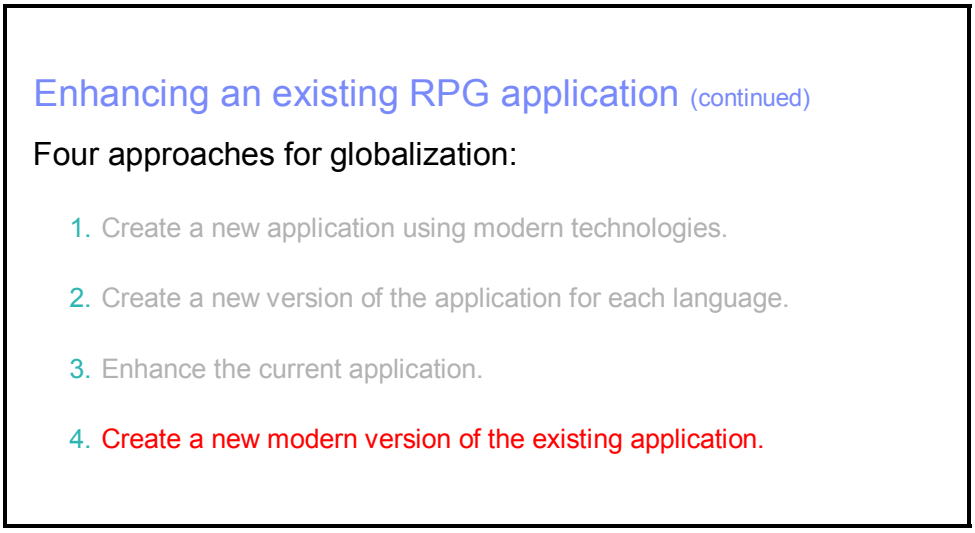
1. Create a new application using modern technologies.
2. **Create a new version of the application for each language.**
3. Enhance the current application.
4. Create a new modern version of the existing application.

Enhancing an existing RPG application (continued)

Approach 2: The second approach involves creating a separate version of the application for each language you want to support. This approach allows you to keep the current application intact, but it requires another application that you have modified to handle a specific language. With this approach, your current users can continue without disruption. It also enables support for new languages as requirements arise in a controlled, phased approach. Additionally, if the new language is similar to an existing supported language, there might be only a few code changes to make. The greatest drawback to this approach is the long-term application maintenance costs. Creating and maintaining multiple application versions will increase development, deployment, and support costs. Another concern with this approach is that the data stored by the application cannot be shared across application versions. This means character data fields will require duplicate data columns (one for each supported language). This results in a more complex database.

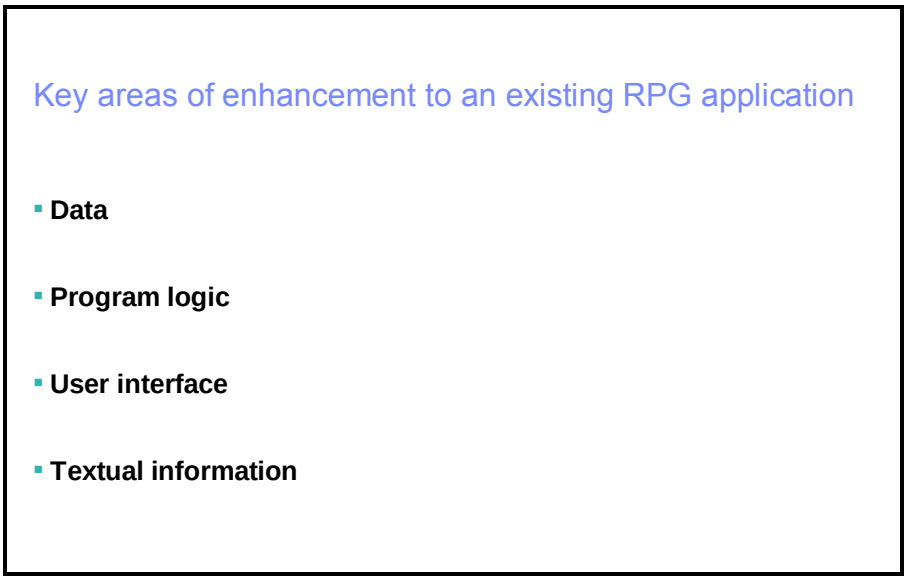
Enhancing an existing RPG application

Approach 3: The third approach involves enhancing the current application. With this approach, you will continue to have just one application and one set of data. As new global requirements arise, you can modify the application as necessary. Over time, you can transform the existing application to be completely global. This approach has many benefits. First, you only have to maintain one application version and one database. Secondly, you can implement globalization changes gradually. This is a practical, recommended approach.



Enhancing an existing RPG application

Approach 4: The final approach allows the existing application and data to remain mostly unchanged for current users, and at the same time, supports the creation of new applications or enhancements that are global. You can take advantage of the existing code base by creating modules that modern languages can call. You can invest your time in writing new code in a language such as Java. You can also add new data in the Unicode format. This encoding standard is one that most solution providers are beginning to use today.



Key areas of enhancement to an existing RPG application

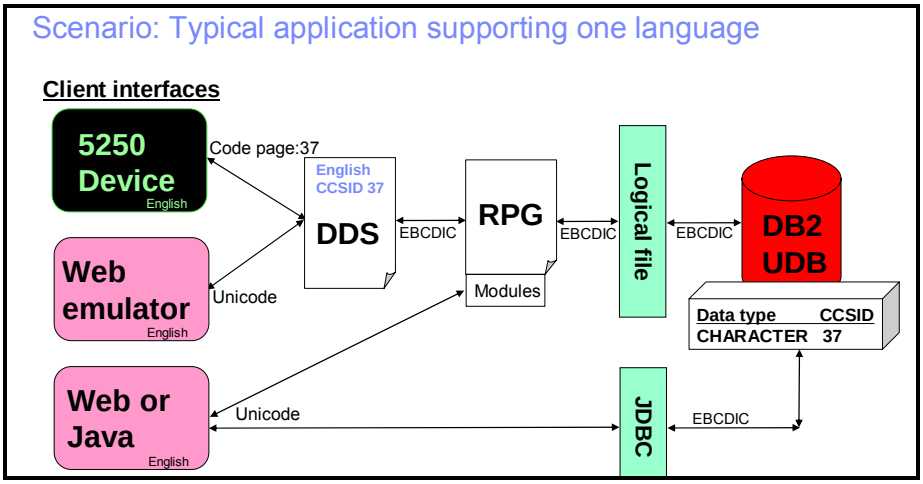
Because many solution providers want to enhance their current RPG applications, here is an overview of the key technical areas on which you will need to focus. At a high level, there are three major focus areas of an application: the data, the core program logic, and the user interface (UI) and textual information. Each of these areas will require some changes to globalize the application.

Because numbers are the same in different languages, the focus here is on character data. Applications must accept the data that a user stores and displays in different languages. Making the data truly global will require changing the CCSID specified for character (SBCS) and graphic (DBCS) data types. Logical views of the data will also need changes. You will see examples of how to do this in future charts.

The core program logic will also require changes. Specifically, you will need to convert the character-related data types to graphic data types. Additionally, you must change every place in the code that references internal variables or strings that relate to the character and graphic database fields. You will see examples of this in future charts.

The UI and textual information must change to display the updated data types properly. Language versions of the DDS and other textual information will be created as part of the translation process.

There are many changes to consider. To help you understand these changes, look at the following scenario that will demonstrate how to approach making these recommended changes.



Scenario: Typical application supporting one language

This diagram illustrates a typical RPG application that supports one language. It addresses three key areas of an application:

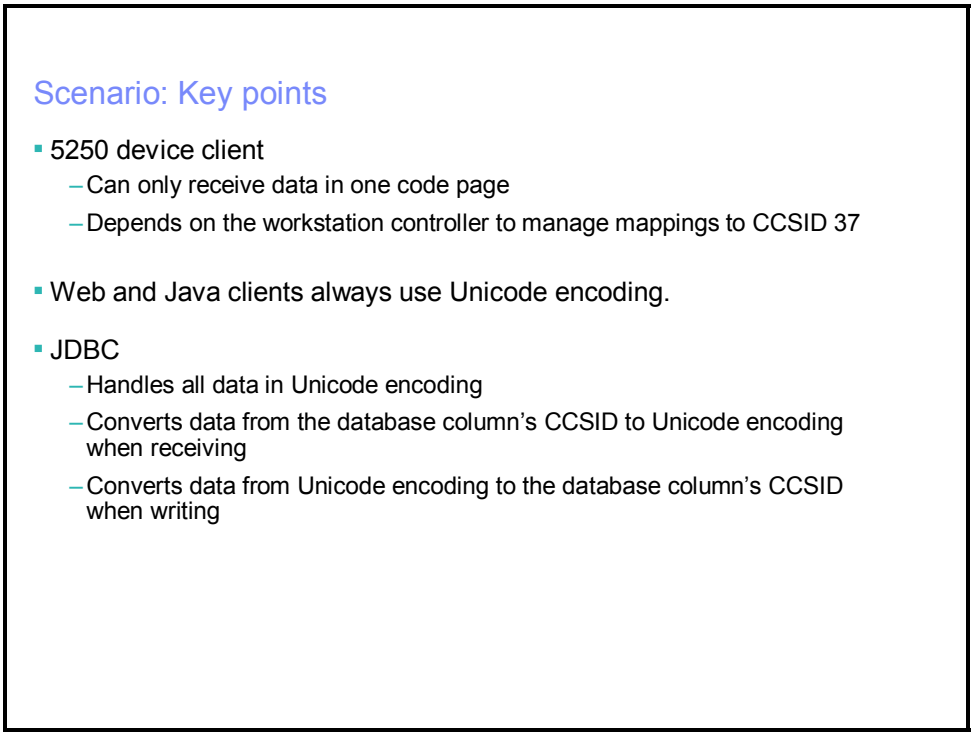
- User interface (UI) (shown on the left),
- Program logic (shown in the middle)
- Database (shown on the right).

All client interfaces see the application UI in English. The supported data contains characters in CCSID 37, which is the code page for English and a few other languages. DDS defines the data fields as CHARACTER data type with a CCSID of 37. The columns in IBM DB2 Universal Database™ (DB2® UDB) for iSeries are also defined as CHARACTER data type with CCSID 37. You can code the RPG code and logical file to handle all data with EBCDIC encoding.

IBM 5250 device: You must configure a 5250 device with a specific code page that dictates which language set of characters to show. Each user selects one code page. In this example, the user chose English code page of 37. The DDS data fields are also tagged with CCSID 37, so the application UI displays in English. When the RPG program receives the input from the UI, the data is in EBCDIC. It is very common to use a logical file or Structured Query Language (SQL) view to look at and update data in a database. Because the character data in the database is defined as CHARACTER with an English CCSID 37, there is no need for data conversions and no risk of data loss.

Web emulator: This type of Web browser accesses the RPG application. You can create these browser interfaces with the IBM WebFacing Tool, IBM WebSphere® Host Access Transformation Services (HATS), or iSeries Access for Web.Browsers always handle Unicode data, so the user can enter this type of encoded data automatically. However, because DDS controls the data fields and expects to see the data encoded in CCSID 37, an error will occur after sending the browser page. The rest of this client scenario is the same for both Web emulator and 5250 device users. The data is handled in CCSID 37 and passed along to the database for storage.

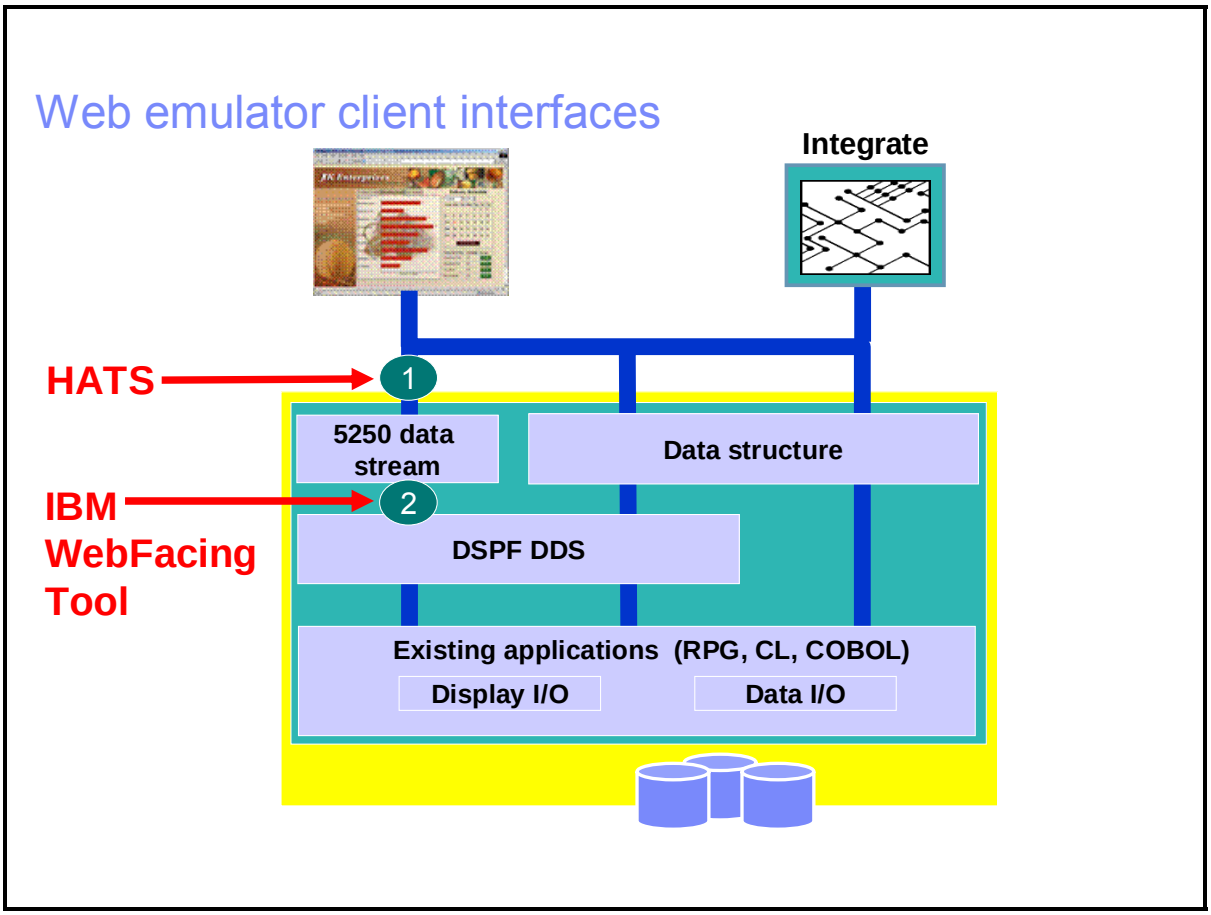
Web or Java: The data in a Java application (a Web application or rich client) always uses Unicode encoding. Java code can call RPG modules directly by using program call markup language (PCML) or stored procedures. In this case, the RPG modules need to handle Unicode data. These same applications can access the database directly through Java database connectivity (JDBC). The JDBC driver automatically converts data. When going from the Java client to the database, the Java client sends Unicode data; JDBC converts the data to the correct CCSID encoding, based on the database column definition where it will be stored.



Scenario: Key points

With 5250 devices, the user must specify and use data according to one code page encoding technique. The workstation controller manages this. Web and Java clients always use Unicode data. Therefore, they do not have limitations on the data.

JDBC has built-in support to handle data conversions between the application and the actual database automatically as needed.

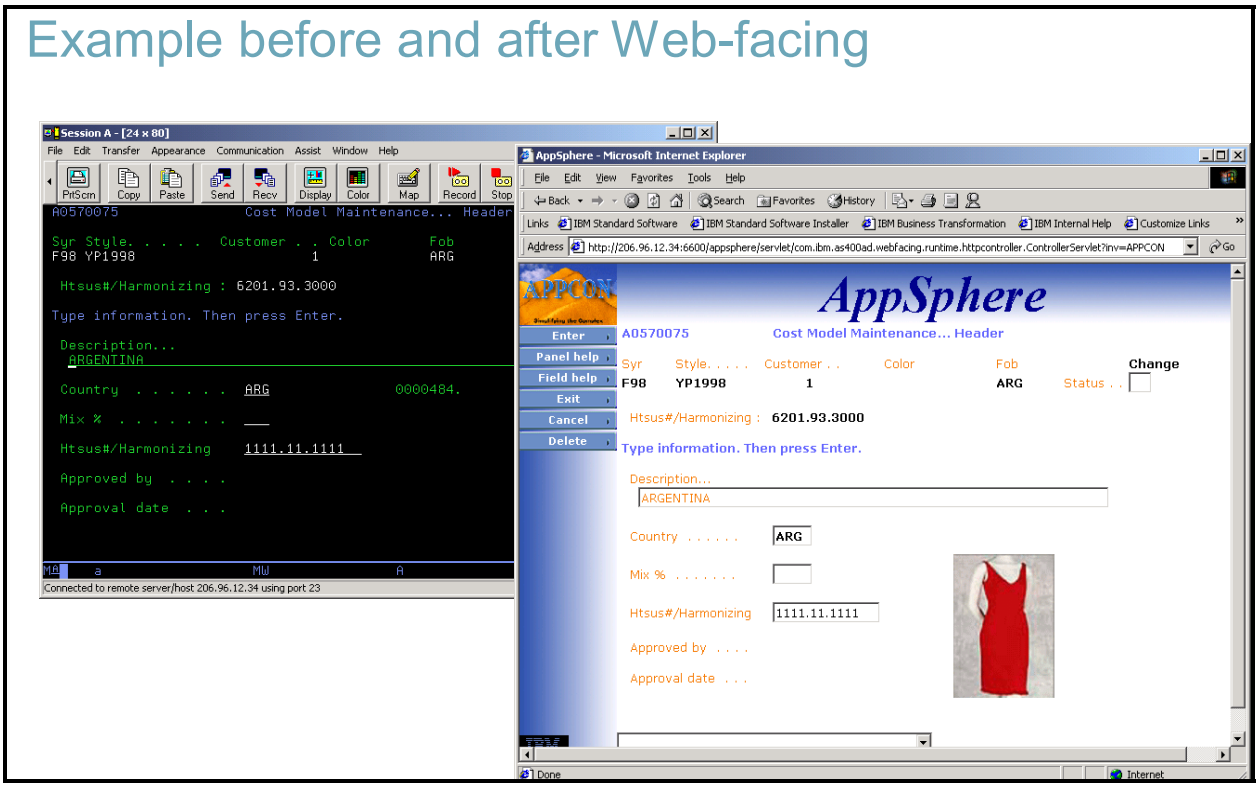


Web emulator client interfaces

This chart shows two types of Web emulation clients; a different IBM tool creates each one. Both provide a Web-based interface that is built from DDS display files. Because all Web browsers support the Unicode Standard, both products also support Unicode Standard.

One of these products is the IBM WebFacing Tool. It works between the application and the output device by converting the interface information to a Web interface and changing the data to Unicode encoding. The IBM WebFacing Tool works off the same DDS code without requiring changes to the DDS. The IBM WebFacing Tool converts the DDS source presentation file (DSPF) to JavaServer™ Pages (JSPs™) and Java servlets during a compile step. This occurs before the 5250 data stream is created.

Another type of Web emulator is Host Access Transformation Server (HATS). It converts the 5250 data stream to HTML dynamically at run time. HATS provides a configuration option called **Enable Unicode Data Stream** that specifies it can accept Unicode data from the application.



Example before and after Web-facing

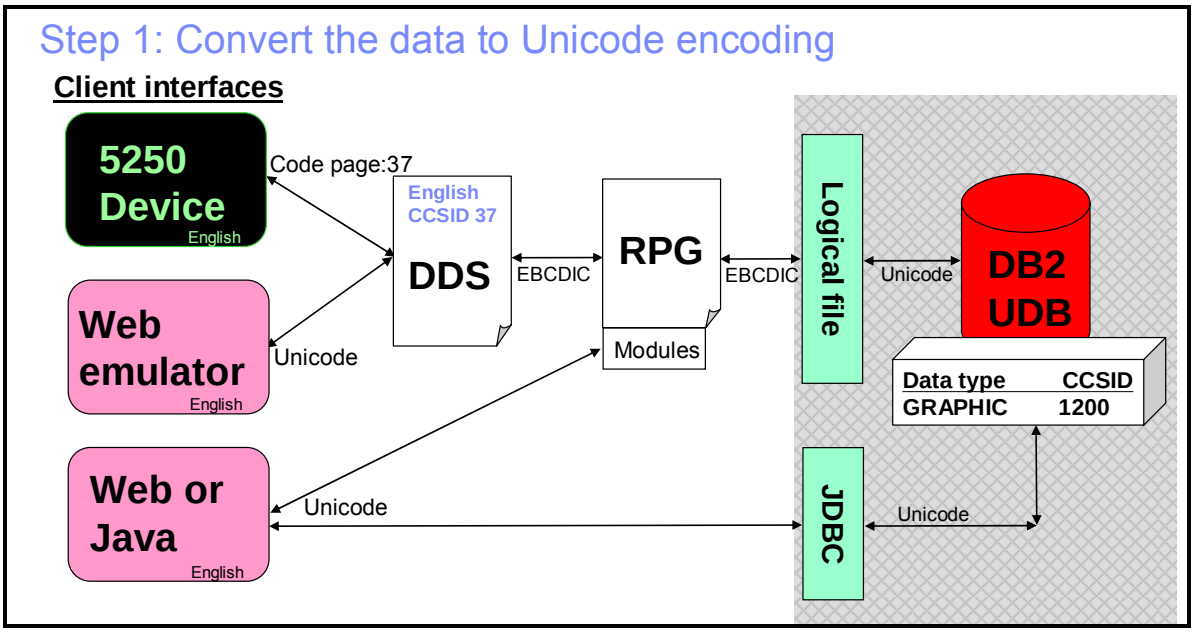
These screen captures show a before-and-after Web-faced green screen. These screens are courtesy of APPCON, an IBM iSeries Business Partner. As you can see, the IBM WebFacing Tool converts 5250 interfaces to browser-based graphical user interfaces (GUIs). With little or no modification to your original iSeries applications, you can extend the use of your programs to the Internet or an intranet. Whether your applications are new or were written before the Internet became a viable platform for conducting business, with the IBM WebFacing Tool, your applications can be available anywhere that users have access to a browser. You can use the IBM WebFacing Tool with applications that invoke DDS source code to create 5250 display screens. The tool has wizards that facilitate selecting your original application's DDS source, converting the source, and deploying the new browser-based interface to your program as an application that works with the WebSphere Application Server.

The conversion creates JavaServer Pages (JSPs) and JavaBeans™ that substitute for your DDS code and make Web access possible. After you convert the DDS code, you can access the application through a browser or continue to use 5250 displays. Having the interface to your applications based on JSPs allows for more flexibility in customizing their appearance. Before the tool converts your DDS code, you can use the Style properties pages to change the look and feel of the generated pages. Styles allow you to define attributes in your Web pages (such as graphics, fonts, colors, and layouts). You can use one of the supplied styles or create your own. To update the appearance of a previously converted project, simply run the IBM WebFacing Tool again and select a new style.

Refer to the **Resources** section for more information about APPCON.

IBM iSeries Access for Web client interface

The third type of Web emulator product from IBM is iSeries Access for Web. This client interface also runs in a Web browser and allows the user to initiate i5/OS applications from it. The iSeries Access for Web tool supports the Unicode Standard. Here is a simple example of a Web-rendered green screen that shows two output fields, each in a different language.



Step 1: Convert the data to Unicode encoding

Now that you have a better idea of the different client interfaces in the typical RPG scenario, look at the first step in enhancing the RPG application to be global. It converts database data to Unicode encoding. The diagram elements in the gray background represent the areas that require change during this step.

All character and graphic database columns are candidates for conversion to Unicode encoding. You can do this by changing their CCSID value to 1200, which is the Unicode UTF-16 format. (UTF-16 is the default Unicode encoding format for the System i architecture). In order to specify the CCSID 1200, the data type must be GRAPHIC (or VARGRAPHIC); you must change this as well. You must change the logical files (that the application uses for accessing the database) to set the data type for these fields, as the data type is different coming in than going out. JDBC automatically handles conversions, so no changes are required there. Now that the data coming from the database and going to the Web or Java applications is all Unicode encoding, these applications require no conversions or enhancements. The next few charts discuss Unicode data in more detail.

Everything else in the scenario remains the same; DDS and RPG programs remain unchanged. When this step is complete, you have a fully functioning, Unicode environment. This step is ideal for solution providers who have both Java and RPG applications accessing the same data.

The Unicode Standard

It provides a means of doing the following:

- Sets the industry standard as a platform-independent way of handling data
- Contains a comprehensive set of characters from all written languages
- Enables a global set of users
- Allows for the exchange of multilingual data with other software or systems
- Simplifies software code

The Unicode Standard

At this point, you have only changed the data to Unicode encoding. The next step will be to change the RPG and DDS code to support it as well.

The Unicode Standard defines a comprehensive set of unique numbers that represent or map to particular characters from all well known, written languages. It also supports scripts, which are sets of characters such as those found in Latin, Greek, and Cyrillic. The Unicode Standard even supports Chinese, Japanese, and Korean ideographs. The unique numbers are the same, regardless of the user's software platform or operating system. Because the Unicode Standard includes all language characters, one field can include characters from many languages. This is called multilingual support.

Unicode encoding eliminates the risk of data corruption because the data does not need re-encoding as other software or systems use it. Data corruption existed prior to the Unicode encoding when software mistakenly encoded data using the wrong encoding system. (You can see an example of this in Appendix A.) This is because the various encoding systems sometimes used the same number to represent different characters. Unicode encoding allows you to specify values for a given field and save it in a mix of languages. Unicode encoding also lets you transport data through many types of systems without corruption, because you no longer have to hardcode whether a value is of a specific language.

Unicode encoding greatly simplifies the complexity and amount of code in a software application when you need to support more than one language, because it provides one encoding system. There is no need to hardcode the encodings for specific languages. Thus, enabling software to handle the Unicode Standard provides support for all languages. Prior to the establishment of the Unicode Standard, the implementation method for encoding systems did not allow support for enough characters. This resulted in the creation of hundreds of encoding systems. When software applications needed to support more than one encoding system, an application had to handle multiple code pages (one for each supported language) to properly handle values in different languages.

With the Unicode Standard, there is also a lesser need for data processing because it is not necessary to encode and de-encode data to accept different languages properly. However, until all parts of the application handle Unicode encoding, some conversions are required.

Unicode encoding or DBCS

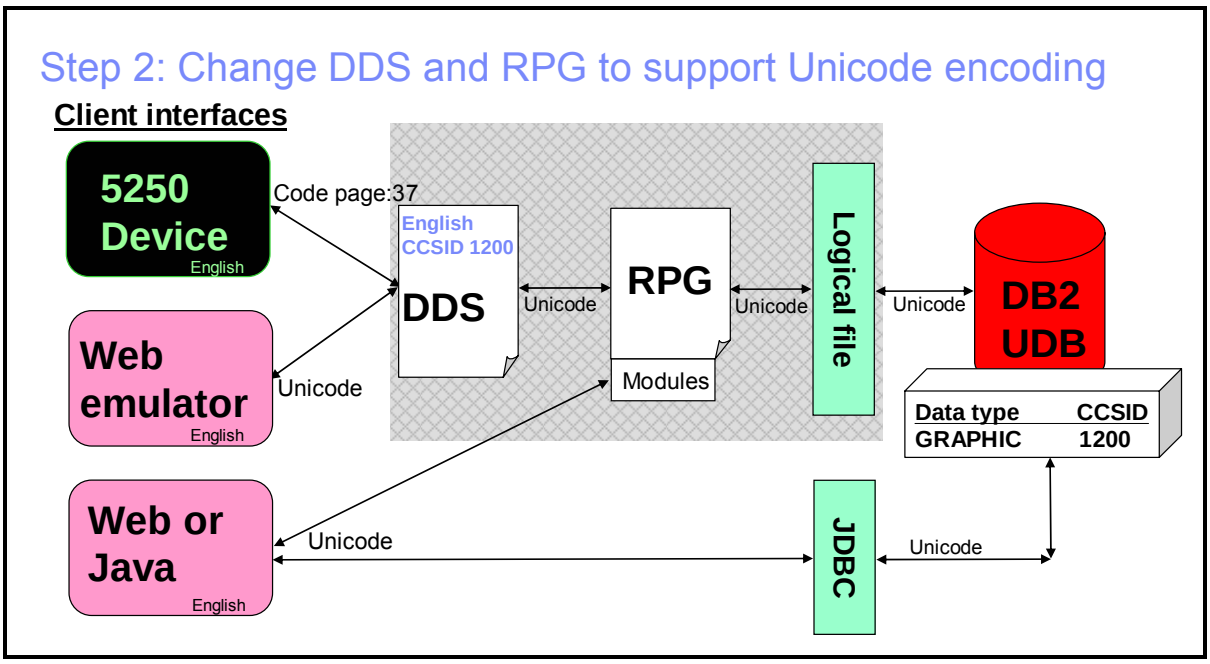
- The strategic direction of the industry and IBM includes:
  - All Web processing is based on the Unicode Standard.
  - More recently developed applications are defined exclusively in Unicode encoding.
- The Chinese government requires software to use Unicode encoding.
- The Unicode Standard lets one application version support all languages.

Unicode encoding or DBCS

Because the issue of multiple language support within software became so important and complex, the Unicode Consortium, an organization sponsored by many companies, created the Unicode Standard. Because the Unicode Standard has been developing over the last few years, the i5/OS platform has not always supported it. In recent releases, Unicode support has grown and is now very functional. IBM wants to ensure that the i5/OS environment is a key platform for running applications that support Unicode encoding in addition to EBCDIC encoding. You can run a mix of both types without either one affecting the other. IBM has enabled i5/OS and other key software for Unicode encoding, including middleware products such as WebSphere and IBM Lotus® Domino® software suites. DB2 UDB for iSeries supports a Unicode data type. The integrated file system (IFS) directory structure has always been based on the Unicode Standard. The data you store in IFS can be global because it is enabled for Unicode encoding.

Aside from the industry direction for Unicode support, there are technical reasons for using this standard. In addition to simplifying developers' coding efforts, the Unicode Standard allows one version of an application to support data in all supported languages. This is positive from the user perspective because they can enter data in their own language without worrying about which characters they can and cannot use. From a programmer's perspective, enhancing a program to support Unicode encoding is about the same amount of work as supporting DBCS encoding. When DBCS-enabling your application, you are supporting only one additional language because you are merely tagging the data in the database with a specific language CCSID. This means that you will need multiple database columns to support multiple languages. Your database becomes more complex and your application code must know which database columns to use.

Because the industry is moving toward the Unicode Standard, and because converting to Unicode encoding requires about the same amount of work as converting to DBCS, it makes more sense to support all languages in one field than to have language-specific fields. There is no reason to support DBCS now that Unicode support is fully available.



Step 2: Change DDS and RPG to support Unicode encoding

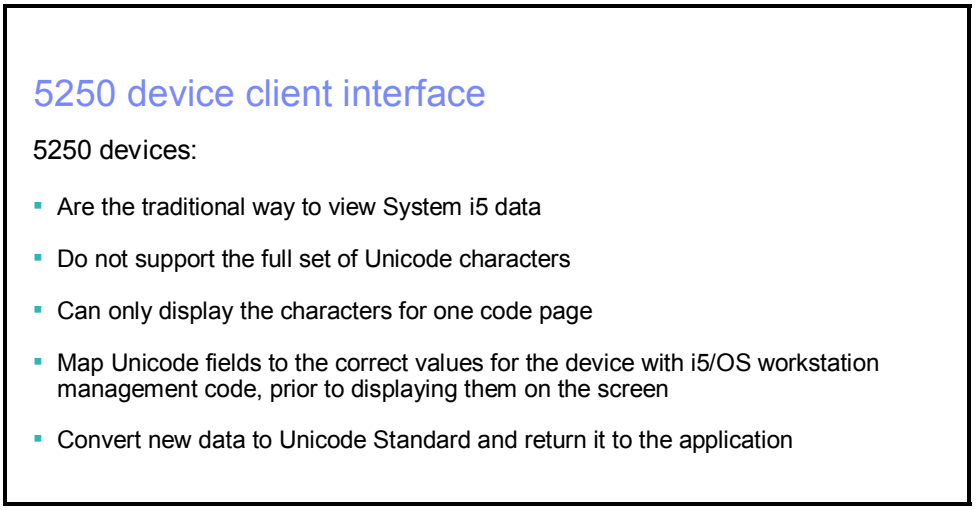
This diagram shows the next typical area of change to support Unicode encoding. This stage focuses on making all the code changes. After these are complete, the environment will handle all character fields as multilingual because the application and interface now supports Unicode encoding.

DDS: There are a couple of different changes to the DDS code. To match the database columns, you need to change each corresponding character field declaration in DDS from **A** (or blank) to **G**, and you need to add a CCSID **1200** value. The combination of **G** and CCSID **1200** define the field as Unicode data in the UTF-16 format. The Unicode Standard requires two bytes to store every character rather than one byte in single-byte CCSIDs. Because the field lengths that the DDS generates to show to a user appear in bytes, these Unicode Standard data fields will now allow the user to enter twice as many bytes. The number of characters in the field is the same, but the 5250 device and Web emulators will now allow twice as many bytes. If needed, you can set a DDS keyword to stop the user from typing too many characters. It is important to update the DDS to support Unicode encoding before making a new language version of the file for translation purposes.

RPG: RPG requires several kinds of code changes to support Unicode encoding. Change the fields that need to be altered from **A** (alphanumeric) to **C**. Next, identify places within the code that compare this Unicode data to EBCDIC. These are usually constant values or temporary internal EBCDIC variables.

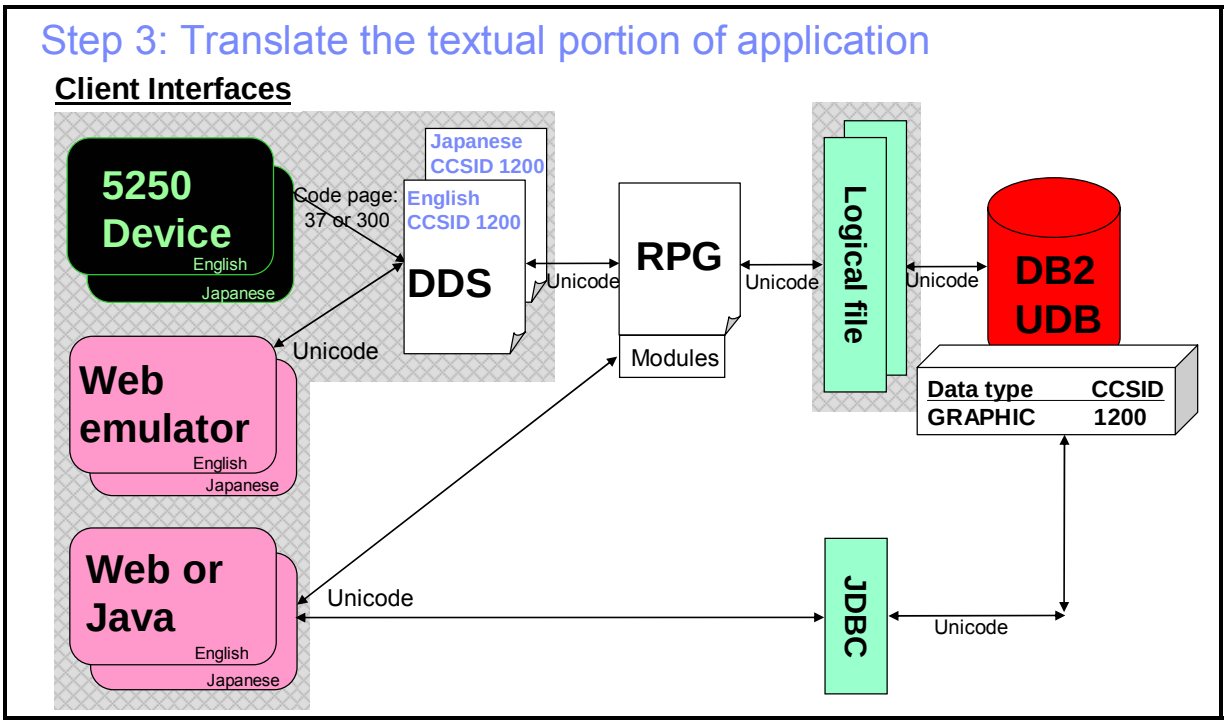
Logical file: The logical file no longer has to support the conversion of data from Unicode encoding to EBCDIC, so you can remove the settings made in Step1.

In this scenario, all the data passes in Unicode encoding without requiring data conversion. The only remaining limitation is that, with the 5250 device, users might not see some Unicode characters.



5250 device client interface

Display devices that currently support the 5250 data stream do not support Unicode data. Therefore, conversions between the Unicode data and EBCDIC are necessary during input and output. On output, the Unicode data is converted to the CCSID of the device. On input, the data is converted from the device CCSID to the Unicode CCSID. The device CCSID, which the configuration specifies, determines what the Unicode data is converted to. The resulting data will appear differently, depending on the device. The device configuration documentation specifies what happens in cases where a Unicode character does not map directly to a code point in the device CCSID. In some cases, a replacement character is used.



Step 3: Translate the textual portion of application

The final step focuses on translating the textual portion of the application so people who read another language can see the user interface in their own tongue. You first need to decide which additional languages to support.

For 5250 devices, the library list controls which language version appears to the user. The code page controls what characters the data fields can contain.

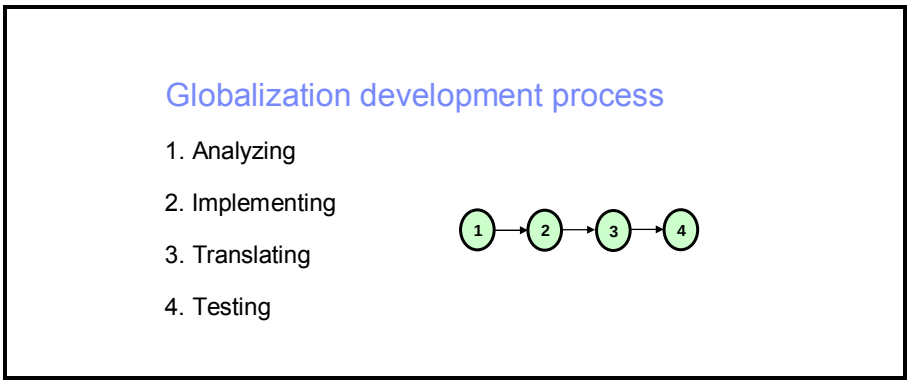
For Web emulators, the library list also controls which language version appears to the user.

For Java applications, the Web browser encoding setting or the user's locale setting controls which language version of the application interface appears to the user.

For RPG, human interpreters translate the required DDS, message files, and other text objects into the target languages.

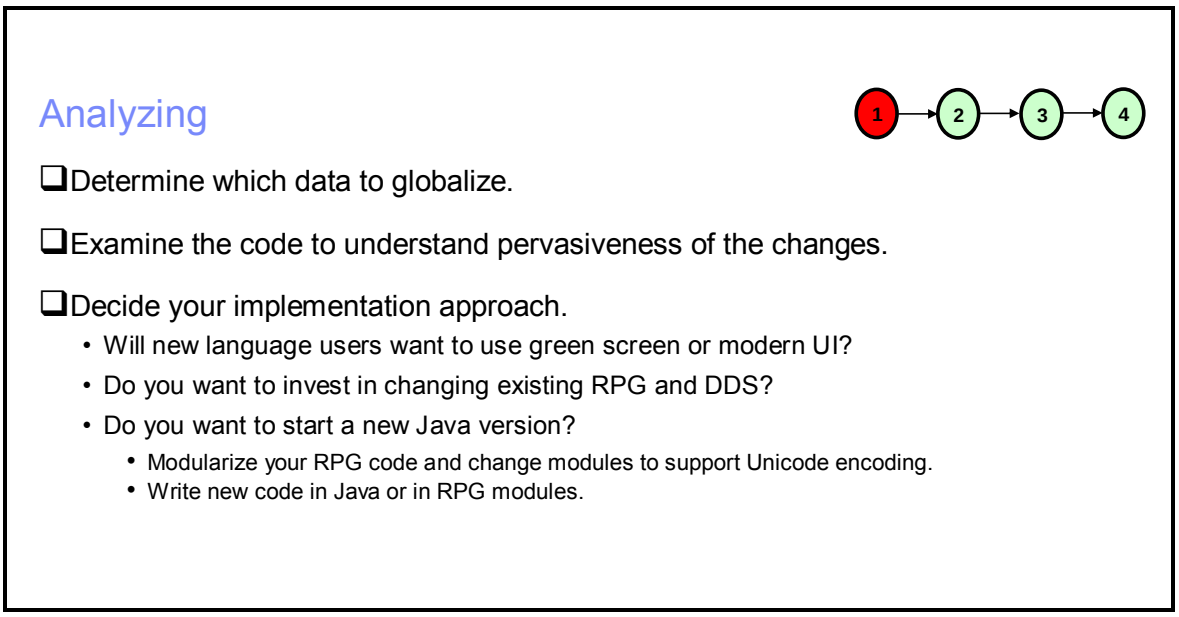
For Java, the human interpreters also translate the contents of resource bundles.

If the culture of the target languages demands the presentation of data in alternate sort orders, you can create logical files with each one defining its own sort sequence. At the completion of this stage, you have an application that not only supports multilingual data, but which is also available for use in more than one language.



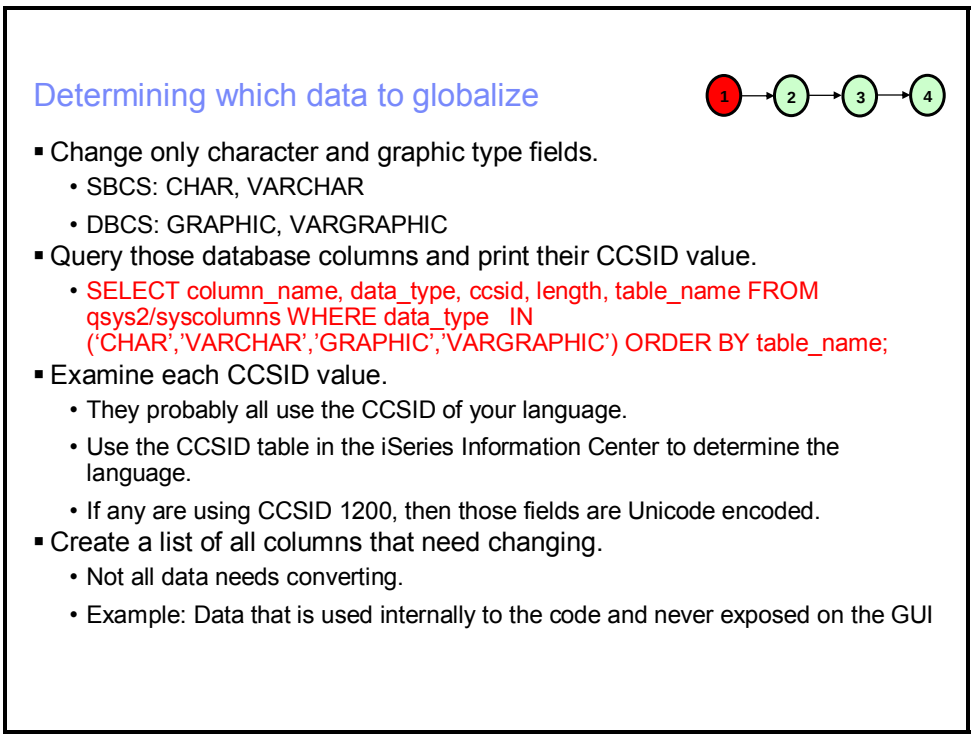
Globalization development process

Now that you have seen how to enhance a typical RPG application to become global, here is a basic process for implementing application changes. Each step will give recommendations, tips, and examples.



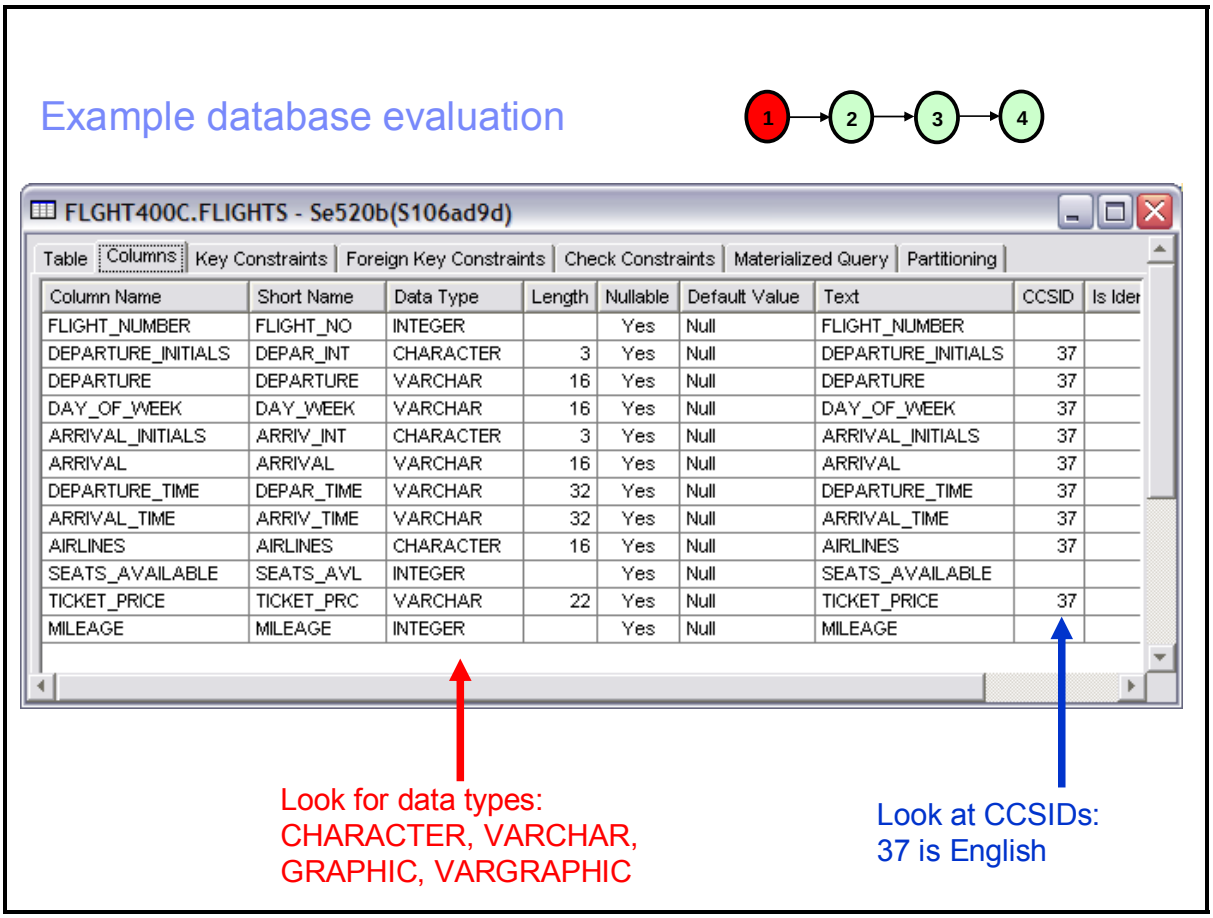
Analyzing

The first step is to determine everything that needs to change, including data and code changes. After you have a good idea of the scope, pervasiveness, and amount of changes necessary, you can decide on an implementation approach. This is the time to consider which user interfaces you want to support.



Determining which data to globalize

When determining which data to globalize, you only need to look at the fields with a data type of **VARCHAR**, **CHARACTER**, **VARGRAPHIC**, and **GRAPHIC**. Accomplish this by running a query of the entire database, searching for columns with those data types, and printing out a list that shows the CCSID for each. (See the sample query on this chart, shown in red.) Look at the CCSID values. If any shows a value of 1200, then that data is already in Unicode format. Other CCSIDs are for specific languages. You can look in the CCSID table in the Globalization section of the iSeries Information Center to see which languages they represent. Start making a list of all the columns. For those columns where the RPG program code uses the data internally, no conversion is necessary. The resulting list will be the database columns you need to convert to the Unicode Standard.

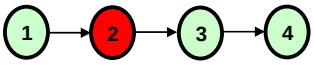


Column Name	Short Name	Data Type	Length	Nullable	Default Value	Text	CCSID	Is Iden
FLIGHT_NUMBER	FLIGHT_NO	INTEGER		Yes	Null	FLIGHT_NUMBER		
DEPARTURE_INITIALS	DEPAR_INT	CHARACTER	3	Yes	Null	DEPARTURE_INITIALS	37	
DEPARTURE	DEPARTURE	VARCHAR	16	Yes	Null	DEPARTURE	37	
DAY_OF_WEEK	DAY_WEEK	VARCHAR	16	Yes	Null	DAY_OF_WEEK	37	
ARRIVAL_INITIALS	ARRIV_INT	CHARACTER	3	Yes	Null	ARRIVAL_INITIALS	37	
ARRIVAL	ARRIVAL	VARCHAR	16	Yes	Null	ARRIVAL	37	
DEPARTURE_TIME	DEPAR_TIME	VARCHAR	32	Yes	Null	DEPARTURE_TIME	37	
ARRIVAL_TIME	ARRIV_TIME	VARCHAR	32	Yes	Null	ARRIVAL_TIME	37	
AIRLINES	AIRLINES	CHARACTER	16	Yes	Null	AIRLINES	37	
SEATS_AVAILABLE	SEATS_AVL	INTEGER		Yes	Null	SEATS_AVAILABLE		
TICKET_PRICE	TICKET_PRC	VARCHAR	22	Yes	Null	TICKET_PRICE	37	
MILEAGE	MILEAGE	INTEGER		Yes	Null	MILEAGE		

Example database evaluation

You can query the database columns through any SQL interface. In this example, the database view in iSeries Navigator is used. Access it by clicking **System** > **Databases** > **Schemas** > **FLIGHT400C** > **Tables**. Choose a table, and then right-click **Definition** to see the database definition for this table. In this example, FLIGHT400C is the database name and FLIGHTS is the table name. You can see that this particular table has many CHARACTER and VARCHAR fields that are candidates to convert to Unicode Standard. All of them support CCSID 37, which is English. At this point, look in the programming code of the application to determine if any of these columns do not need to be converted. For example, if the strings from a character-based data type column are only used internally in the program source code and are never shown on the user interface, this database column does not require conversion.

Implementing



- Convert code to RPG IV.
- Separate text from business logic.
- Convert character data in DB2 UDB to Unicode encoding.
- Make changes in RPG code to support Unicode encoding.
- Make changes in DDS to support Unicode encoding.
- Change install code.

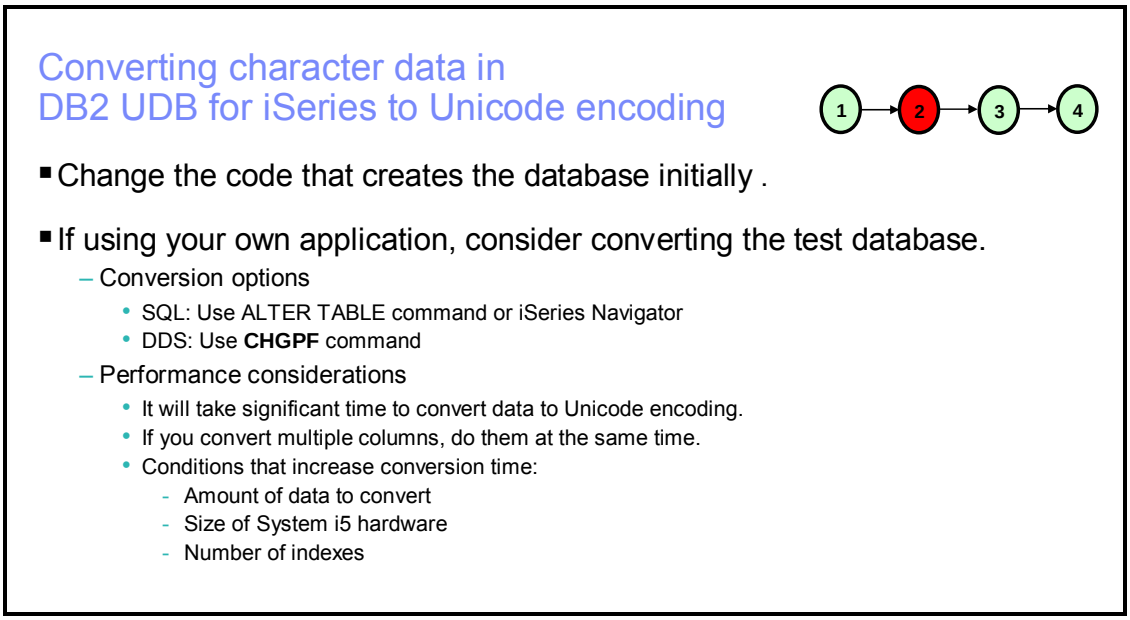
Implementing

In this step, you will implement all of the code changes. If you have a huge application, this step can take many months to complete. The best thing to do is to stage out how to make sets of changes at a time. You might want to support some fields in the Unicode Standard in one particular release of the application and leave other fields for another release.

First, you need to convert your RPG code to RPG IV. This will require a recompile (described in the iSeries Information Center). RPG IV provides a lot of Unicode support, including functions for date and time formatting. Additionally, moving to the Integrated Language Environment for RPG IV (ILE RPG) allows you to create language-specific code modules. Next, ensure that all of your textual information is separate from the core program logic. If you find hard-coded strings that are part of the user interface, move these out of the code and into DDS or message files.

You are now ready to convert your database columns to Unicode encoding and make the Unicode changes to the RPG and DDS code. If there are many changes, determine a plan of action, perhaps program-by-program.

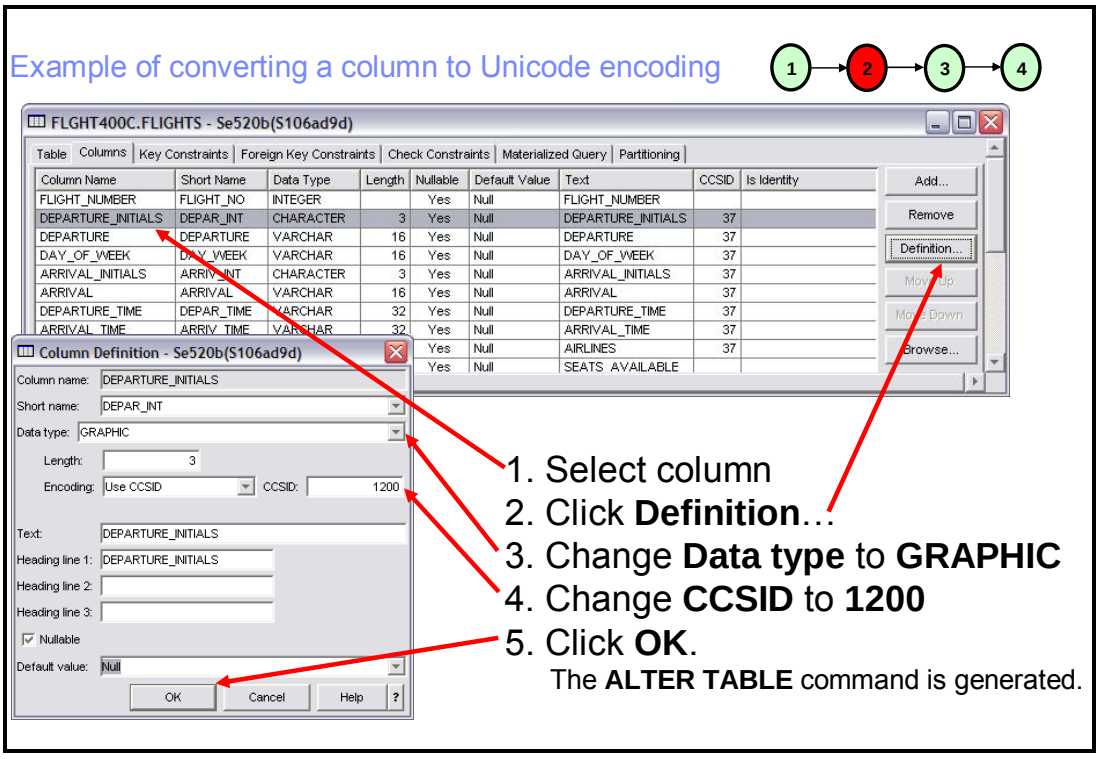
You will also need to look at whether your application's installation code needs updating. When supporting an application in more than one language, you can create separate software CDs for each language, or you can build one installation CD and prompt the user who is installing the application to select which language version to implement. For a current user of your application, you will need to write code to convert the data in their existing DB2 UDB for iSeries database to Unicode encoding. This data conversion will likely run as part of an application upgrade.



Converting character data in DB2 UDB for iSeries to Unicode encoding

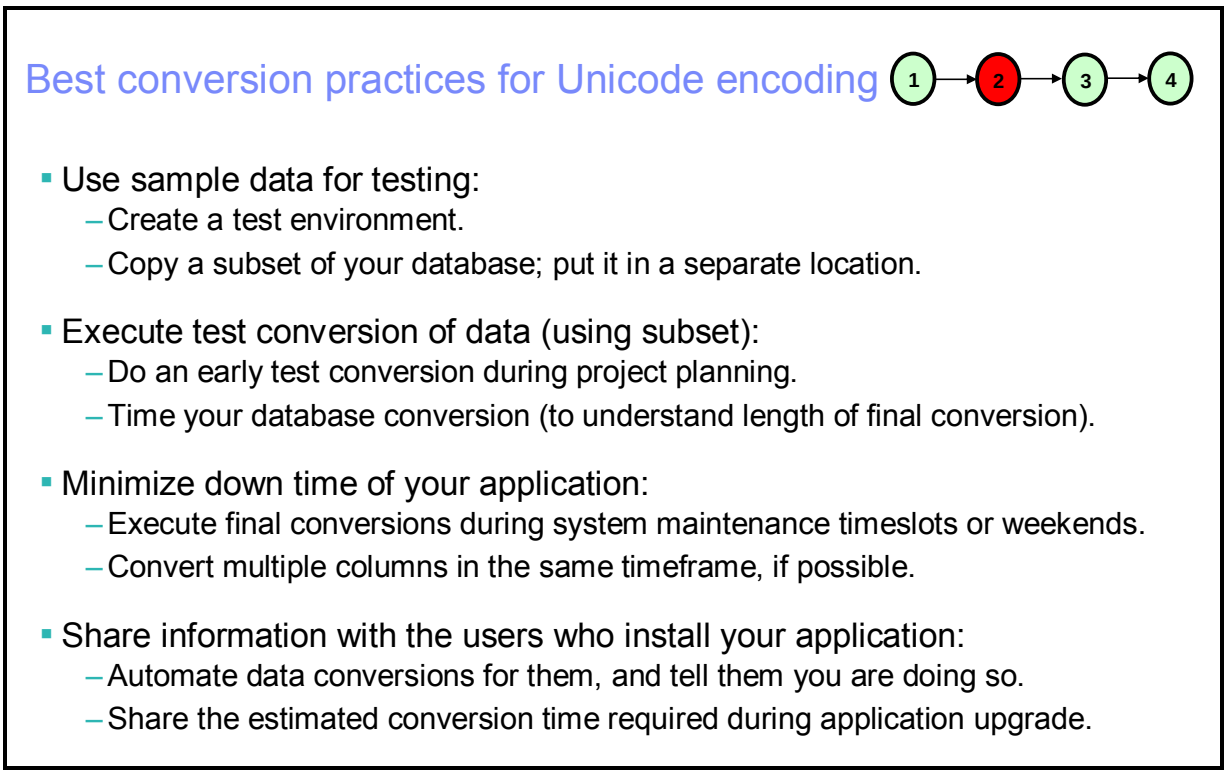
Here is some useful information about converting data to Unicode encoding. You must update any install code that creates a database to build those database columns in Unicode encoding. If your application code is operating in a test or production environment, you must convert those databases, too. Converting a database column from to a Unicode data type is similar to changing the length of a column. You can use the SQL **ALTER TABLE** command or the i5/OS **CHGPF** command.

There are several performance items to consider. It will take quite some time to convert data to Unicode encoding. You can do a test against a typical installed database to estimate how long this will take for your customers. If you need to convert more than one column, it is more efficient to convert them all at once in one command. Three key factors will influence how much time it will take to convert the data: the amount of data to convert, the size of the System i hardware on which it is running, and the number of indexes that need rebuilding.



Example of converting a column to Unicode encoding

You can use the SQL view in the iSeries Navigator to see how to accomplish the conversion. In the top screen shown here, highlight the database column name that requires conversion, and then click the **Definition** button. The small window at the bottom appears. Change the **Data type** to **GRAPHIC** and change the CCSID from **37** to **1200**. Then, click **OK** to generate and run an SQL **ALTER TABLE** command.



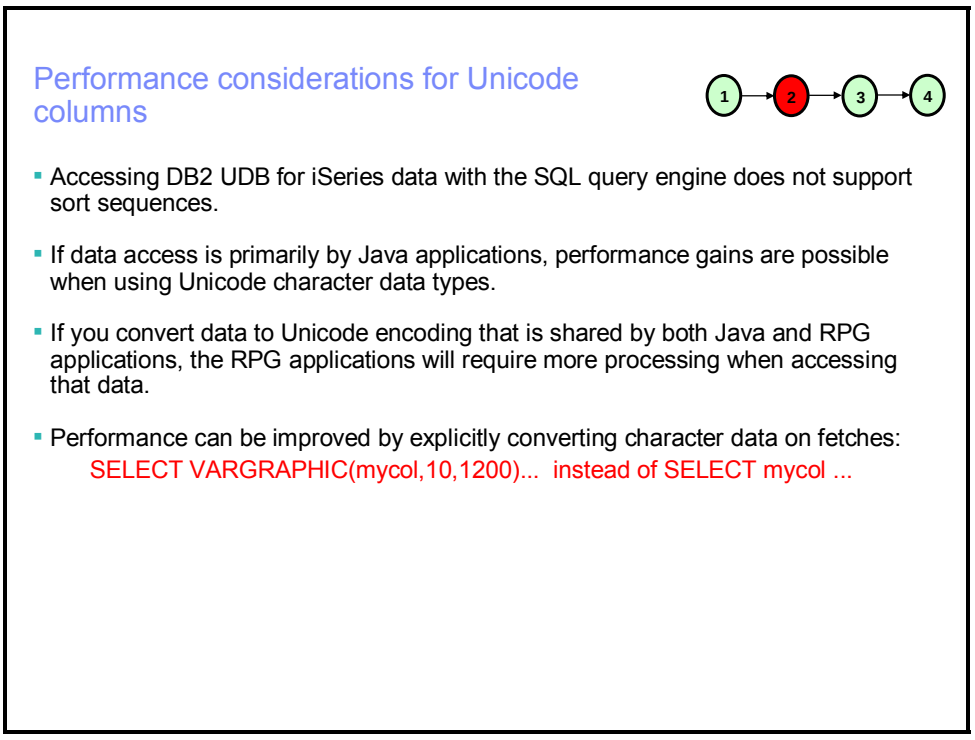
Best conversion practices for Unicode encoding

Some best practices and tips for creating data conversions to Unicode Standard include using sample data for testing your Unicode modifications. You will need to create a test environment in a separate location. Then, copy a subset of your database into this test environment.

You need to test the conversion code you have written, using a subset of the data that is to be converted. It can be valuable to do an early test conversion during the project-planning phase. Be sure to time the conversions so that you can better understand the anticipated length of the final conversion efforts that your user will experience.

It is important to minimize application downtime for all applications that are affected by the introduction of the Unicode encoding. It is probably best to execute final data conversions during system maintenance timeslots or weekends. Convert multiple columns in the same timeframe if possible.

Be sure to keep the users who are installing the Unicode conversion programs aware of what is happening. Display messages about the estimated conversion time required during application upgrades, and automate all data conversions for them.



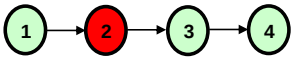
Performance considerations for Unicode columns

When applications use the SQL query engine to access DB2 UDB for iSeries data, there is no support for sort sequences. In fact, attempting to do this can appear as a performance problem.

If more than one application accesses the same data, consider converting the data to Unicode encoding. However, realize that accessing the data from traditional enterprise (non-Java) applications will be slower because of the real-time data conversion.

You can also improve the performance of data conversions by explicitly specifying that the query engine retrieve the data value in Unicode encoding. The example SQL **SELECT** statement (shown in red on this chart) illustrates how you do this.

RPG code changes



- Specify Unicode data as C data type to variable declarations.
- Change code that compares constants with Unicode data.
- Change code that moves Unicode data around by using internal temporary variables.
- Find some required code changes by compiling the RPG code.
- Remove cultural assumptions put into DDS where possible.

RPG code changes

Several types of RPG code changes are necessary to use Unicode encoding.

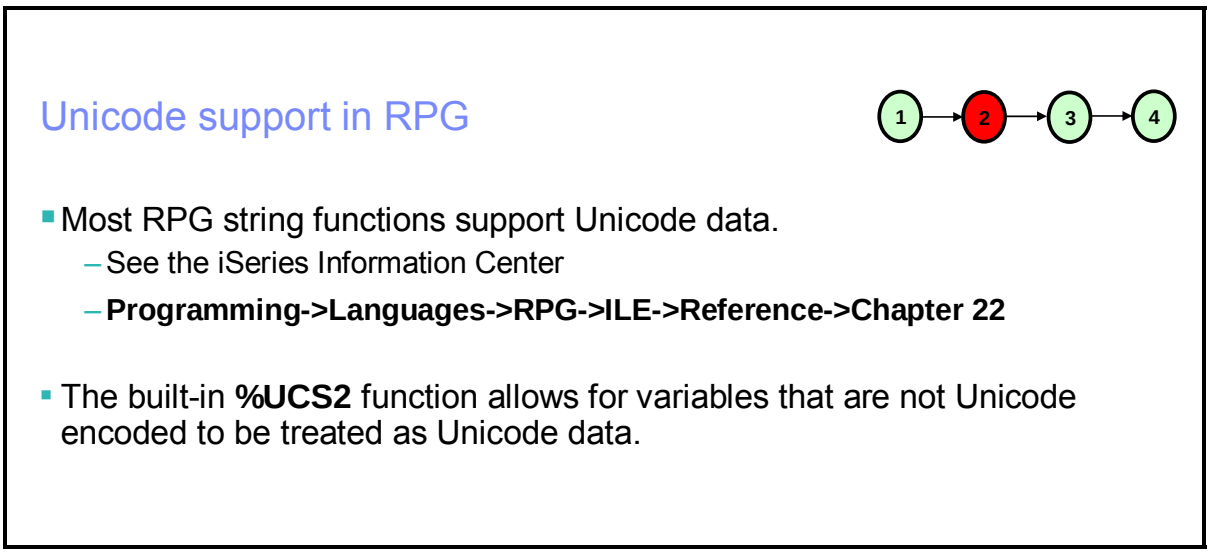
For variable declarations, you must specify Unicode data as a C data type.

Be sure to change code that compares constants with Unicode data.

Modify code that moves Unicode data around by using internal temporary variables.

You can find some required code changes by compiling the RPG code and then looking at the resulting warnings.

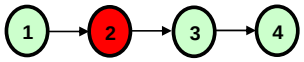
Take into consideration any cultural assumptions within the DDS code, and remove these where possible.



Unicode support in RPG

Many string-related functions provide automatic detection and conversion support between EBCDIC and Unicode fields. Examples of these include the **MOVE**, **MOVEL**, and **EVAL** functions. You can see a complete list of these in Chapter 22 of the RPG Reference, which is online at the iSeries Information Center. (Follow the navigation shown in this chart.) An added built-in function allows you to convert a constant string or variable into Unicode format before using it with Unicode functions or comparisons. The function name is **%USC2**.

Examples of using Unicode encoding in RPG



- You can use the **C** data type and the **%ucs2** function to wrapper an compare an EBCDIC data field with a Unicode (UTF-16) field:

```
H CCSID(*UCS2 : 1200)
 * Initialize a Unicode variable fieldU to a valid Unicode value
D  fieldU         S      10C  INZ(%ucs2('abcdefghij'))
 * Assign the EBCDIC value MyDataE to the Unicode variable MyDataU
C                   eval      MyDataU = %ucs2(MyDataE)
 * Compare an EBCDIC value to a Unicode value
C                   if        MyDataU > %ucs2(MyDataE)
```

- The **movel** and **trimr** functions automatically detect the type of variable (EBCDIC or Unicode data) and handle automatic conversion as needed by the function:

```
 * Converts the constant 'Winter' to Unicode if MyDataU is defined as Unicode
C                   movel     'Winter'      MyDataU
 * Trims blanks off the right side of the string. String length is properly
 * determined to find the string end. Ex. EBCDIC length is 6, Unicode is 12
C                   eval      MyText = %trimr(MyText)
```

Examples of using Unicode encoding in RPG

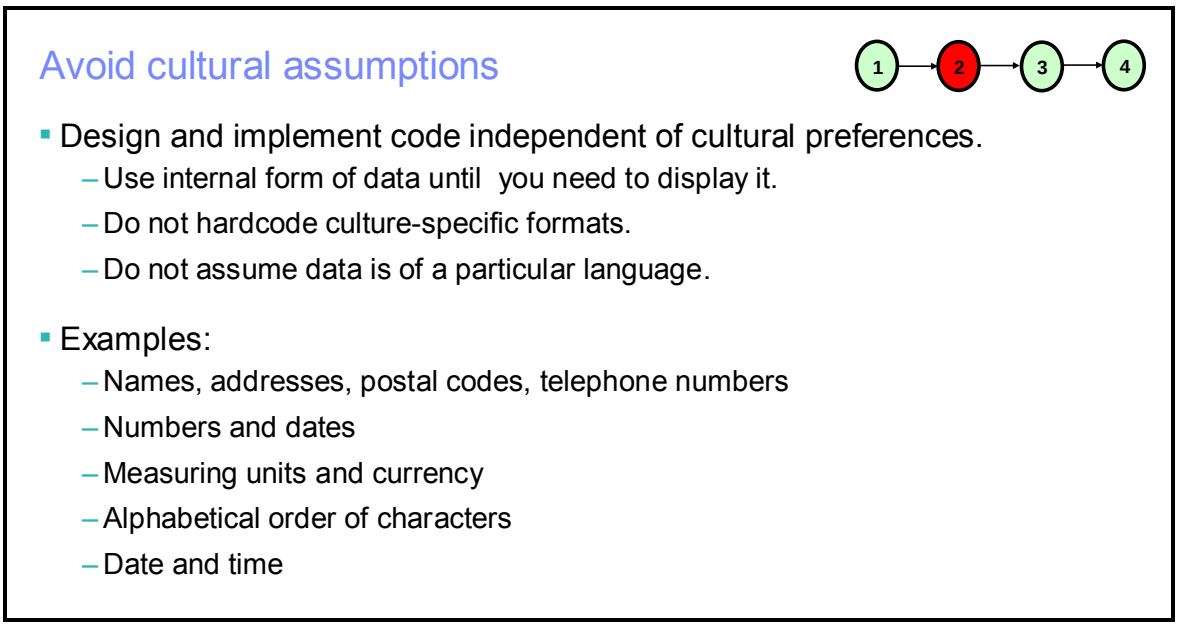
Here are two examples that illustrate how RPG programs support Unicode data.

The top example shows how to define a Unicode field with the **C** data type. It also uses the **%ucs2** function to wrapper an EBCDIC value data field so the value can be stored or compared with the Unicode field.

You can use an **H** spec (Header specification, shown in red) to define the particular Unicode CCSID value for all **%ucs2** invocations in the RPG program. Alternatively, the program can specify the Unicode CCSID value on each call to the **%ucs2** function. For example:

```
%ucs2(MyDataE : 1200)
```

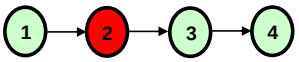
The second example shows two ways to support Unicode encoding. One method uses the **MOVEL** function, and the other method uses the **%trimr** function.



Avoid cultural assumptions

To globalize an application completely, you need to be aware of how you handle data that has cultural significance. You must store the data generically in the database and convert it to the right format just before displaying it. In the code, use the internal generic format until a user needs to see it. Avoid hard coding culture-specific data. However, because RPG does not provide ways to format all culturally significant data, this will be impossible to avoid completely. The list in this chart shows examples of data that requires different formats, depending on the user's cultural preference.

Formatting data in DDS



- Format data in DDS, if possible:
  - Use the **DATFMT** keyword for the date format.
  - For example, use the **DATFMT(*JOB)** keyword in DDS:
    - Uses the job attribute, which is the setting from the user's user profile
    - Provides approximately 10 choices
    - If the required format is not an option, then handle it in your code.

```
00010A
00020A    R RECORD
00030A      DATFLD1       L  B  5  2DATFMT(*JUL)
00040A      DATFLD2       L  B  5 22DATFMT(*EUR)
00050A      DATFLD3       L  B  5 42DATFMT(*JOB)
```

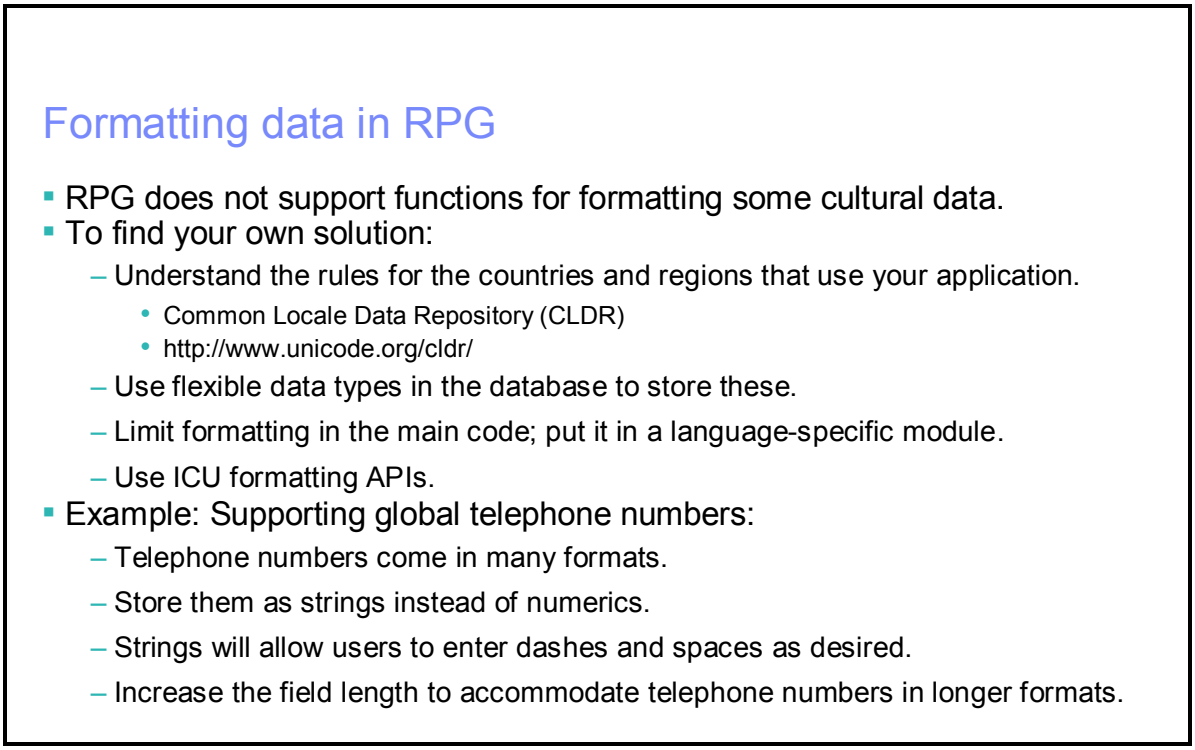
February 1, 2006 equals:

```
DATFLD1   06/032
DATFLD2   01.02.2006
DATFLD3   02/01/06
```

Formatting data in DDS

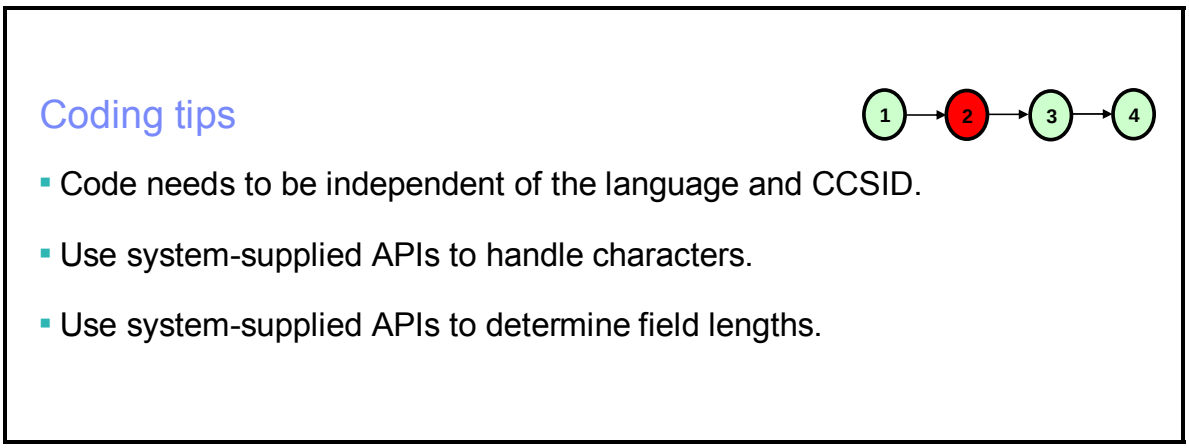
Use built-in formatting keywords in DDS to format data. Each language version of DDS will specify its preferred format. Alternatively, you can use conditional statements in DDS to display one format instead of another.

This chart provides an example of using the date formatting (**DATFMT**) keyword in DDS. The DDS file on the left shows the definition of three data fields, each specifying a different style format: Julian (***JUL**), European (***EUR**), and the format from the date format job attribute. Use the DDS **DATFMT** keyword to specify these. When these formats are applied against the same date of February 1, 2006, the outcome varies. You can see the different formatting results on the right side of this chart.



Formatting data in RPG

DDS only supports a few data formats, which means you will have to handle the other culturally sensitive data formatting in the RPG code. However, RPG also does not provide built-in formatting functions. Thus, you will need to decide how to handle this in your code. First, start by understanding the rules for how the particular country or region formats the kind of data you are handling. You can find some of this information online in the Common Locale Data Repository. The easiest thing to do for flexibility is to use character data types for things such as telephone numbers. This allows flexibility for users to enter parentheses, dashes, and spaces where desired. You might also want to increase the field length to accommodate longer phone numbers. You might need to break language-specific formatting code into separate, callable modules to keep them out of the main language-independent code. You might also want to use the International Components for Unicode (ICU) application programming interfaces (APIs) to help you format data. You can find out more about these APIs at the IBM International Components for Unicode Web site listed in the **Resources** section of this course.



Coding tips

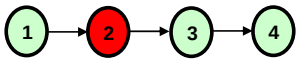
There are three key general coding tips you need to follow as you modify your applications to support Unicode data.

You need to write the code to be independent of any particular language and CCSID. Also, you need to isolate different sets of culturally dependent and text-dependent code into modules. Do not write your code to depend on the specific job CCSID. These steps will reduce the risk of your application generating unpredictable results when running on differently configured systems.

Be sure to use system-supplied APIs to handle character data and field lengths. Do not design your application to depend on specific character values.

Use system-supplied APIs to determine field lengths. This is important because field lengths vary with different encoding schemes. For example, field lengths are the same for SBCS and DBCS; however, a DBCS character requires two bytes.

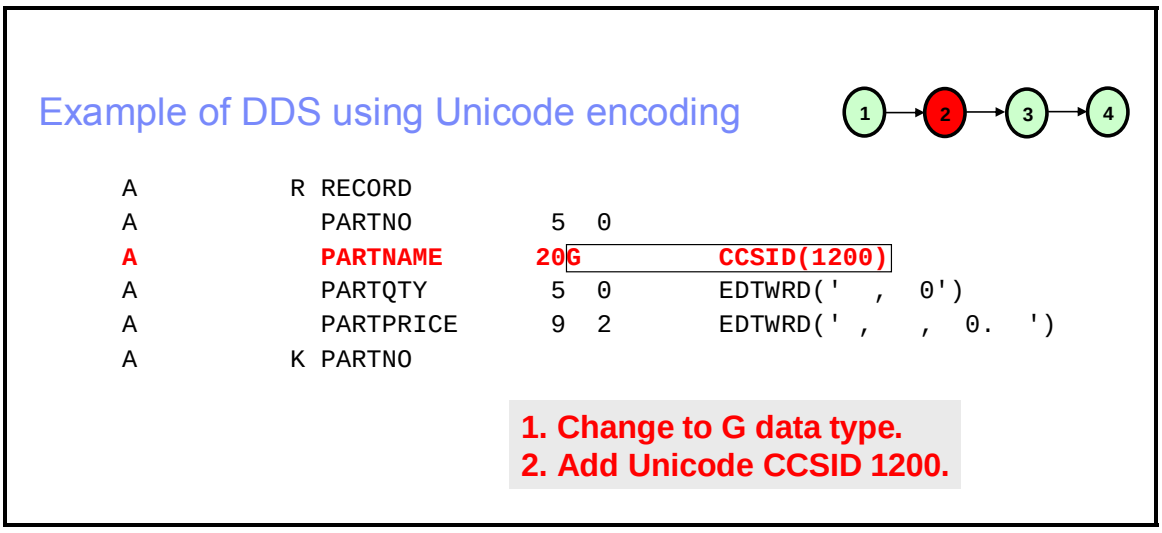
DDS code changes



- Change the data type of alphanumeric fields to Unicode encoding:
  - Change to graphic (G) data type.
  - Add CCSID value 1200 for that data type.
- Calculate the required text expansion for text strings:
  - Adjust layout for each string within the screen to accommodate expansion.

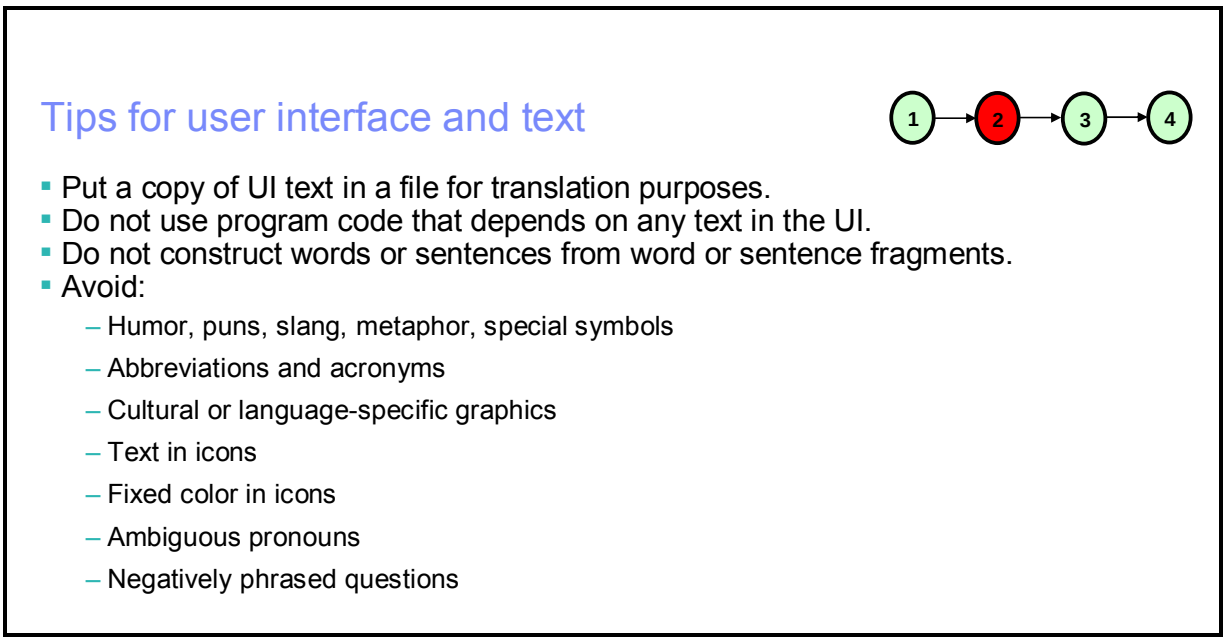
DDS code changes

There are two things to change in the DDS file. First, change the alphanumeric fields to Unicode encoding and specify the CCSID value of **1200**. Together, these values define the data as Unicode encoding. Next, adjust the layout of the strings on the screen to accommodate the extra room needed when translating the strings into another language. Make the adjustments prior to having the DDS translated in order to maintain one master copy of the DDS. You will learn more about this during the translation step.



Example of DDS using Unicode encoding

Here is the Unicode version of the DDS source code. You can see the DDS changes to make PARTNAME a Unicode field. Simply add a **G** with a CCSID value of **1200**. Together, these two elements define the field for Unicode encoding.



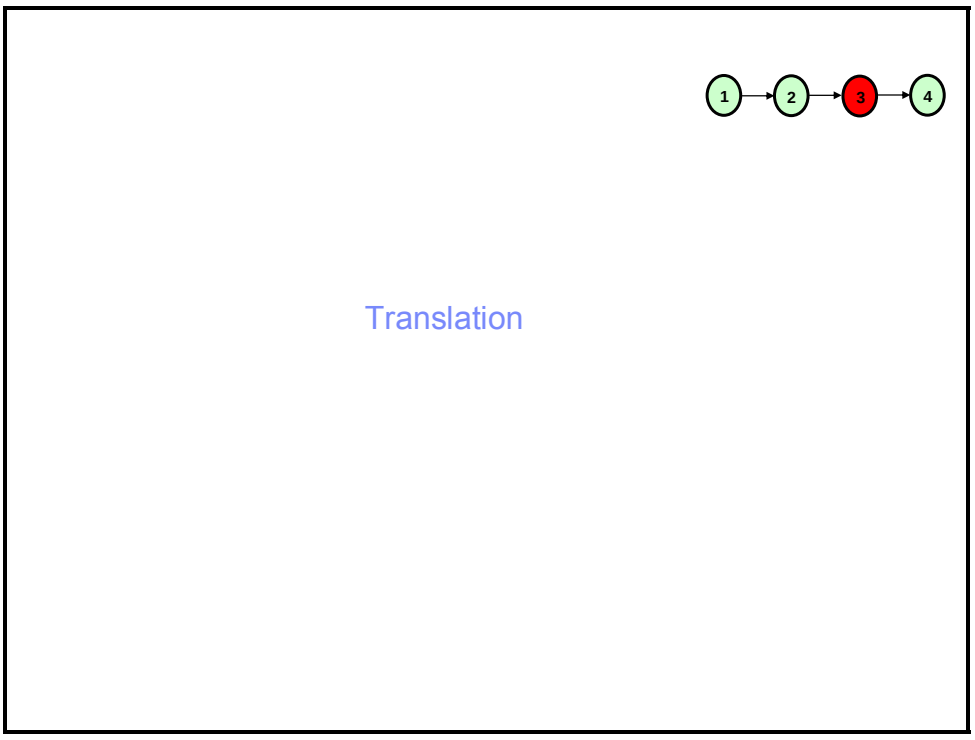
Tips for user interface and text

Here is a set of tips to remove any textual dependence in the code. This improves the quality of the user interface and allows the application to avoid having to handle text that is difficult to translate.

First, consider putting a copy of UI text in a file to allow translation if it is stored in your database. However, be sure that the program code is not dependent on any text that the UI contains. It is best to avoid constructing individual words from word fragments, and similarly, try to avoid constructing a sentence from sentence fragments.

Other things to avoid within text strings include humor, puns, slang, metaphors, and special symbols; these will usually not translate well into other languages. Abbreviations, acronyms, and specific cultural or language-related graphics also translate into other languages poorly, inappropriately, or not at all.

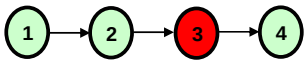
Generally, other application coding techniques can also be challenging to deliver to an international audience. These include text or fixed colors in icons, as well as negatively phrased questions.



Translation

This section discusses the third step in the globalization process: translation.

The translation process



- The translation process converts text written in one language into another language.
- There are several types of software-related textual information:
  - User interface strings
  - Messages, help text
  - Menu and command text
  - Installation interface (wizard)
  - Documents
    - Used by application users, installers, and administrators
- RPG UI text is in files in one library.
- Java is in ResourceBundle.
- There are ywo translation methods:
  1. Machine translation
  2. Manual translation

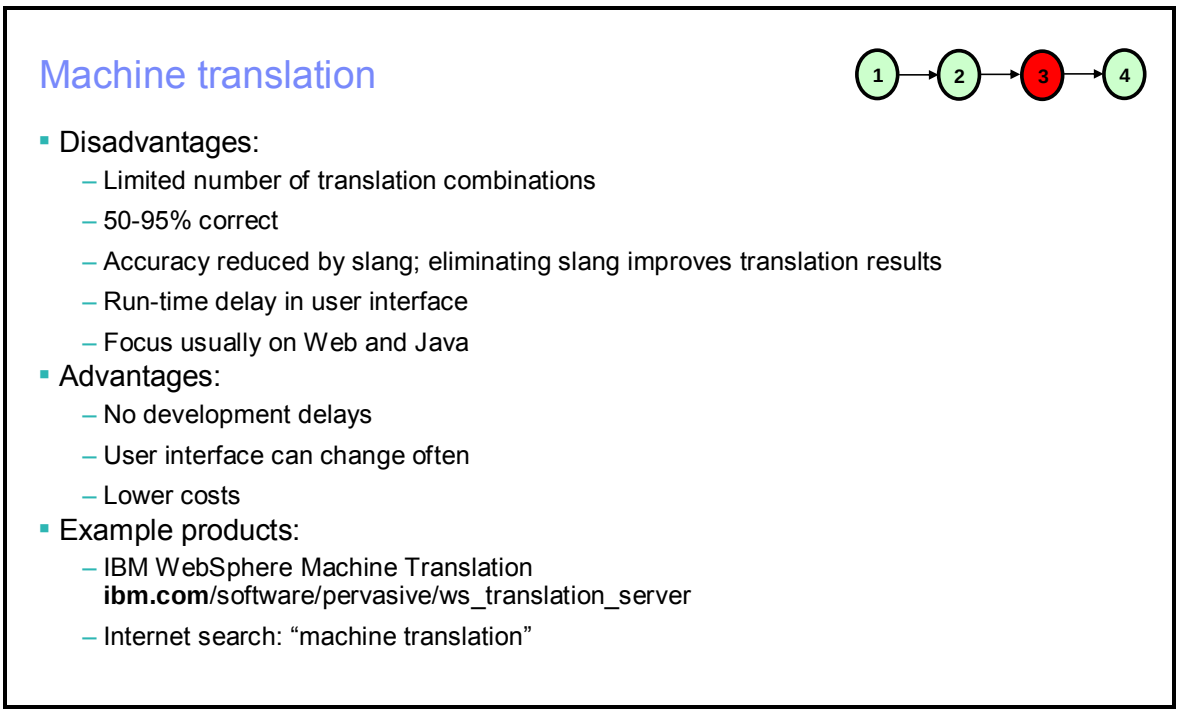
The translation process

Translation is the process of converting text into another language. All textual pieces of the application need to undergo translation, from user interface to the documentation.

The DDS and message files usually contain the UI data for RPG.

For Java, the resources bundles contain the UI data.

There are two methods of translating text: machine translation and manual translation. Machine translation interprets text on demand (dynamically) as the application runs. Manual translation involves human interpreters. Manual translation is the ideal approach, because current machine translation technologies are not yet accurate enough.



Machine translation

There are a few advantages to machine translation, but the many disadvantages outweigh them by far. Because of this, there will be a discussion of the disadvantages first.

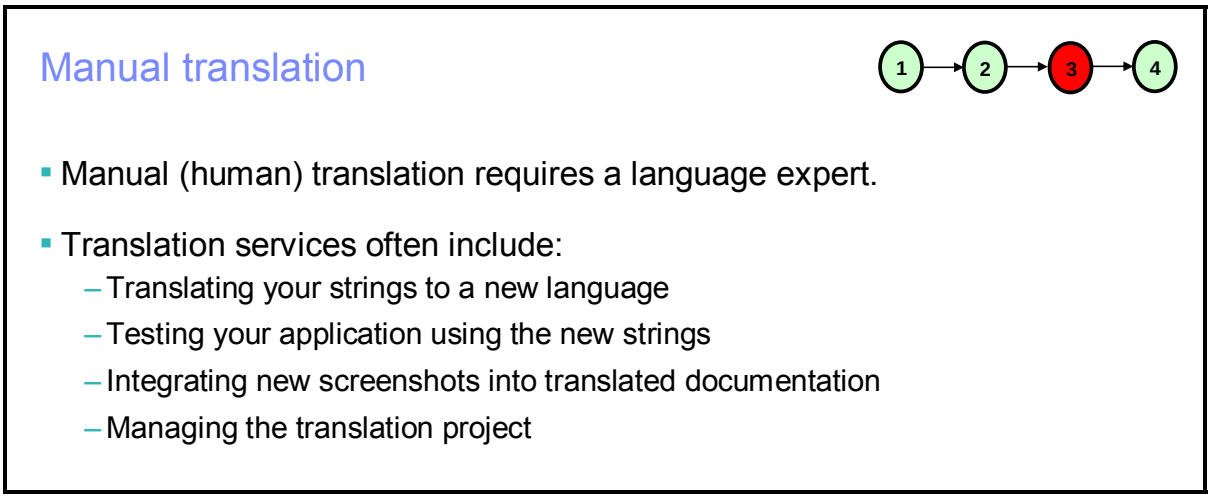
Machine translation limits the number of translation combinations. The resulting translations are rarely even 95% correct, and can often be only 50% correct. The percentage of correct translation results will depend on the simplicity of the sentences. The best sentences will use short, present-tense, active-voice construction with no interjections or prepositional phrases in the middle of the statement. Similarly, quotes (around an emphasized word) and dashes for punctuation can result in poor machine translations.

The use of slang words and colloquialisms, though perhaps clever in the native language, makes automatic translation difficult or impossible. Eliminating slang and simplifying your documentation will improve translation results.

Because machine translation is dynamic, it can cause run-time delays in the user’s experience. Additionally, most machine-translation products focus on Web and Java interfaces, not on the more traditional user interfaces.

However, there are a few advantages to using machine translation. There is no delay during development while waiting for the manual translation to be completed. It is possible to change the user interface often, without going through a manual retranslation process. Lastly, machine translation offers a lower cost solution. The solution provider merely needs to make a one-time product purchase and then can use the machine-translation tool for the life of application.

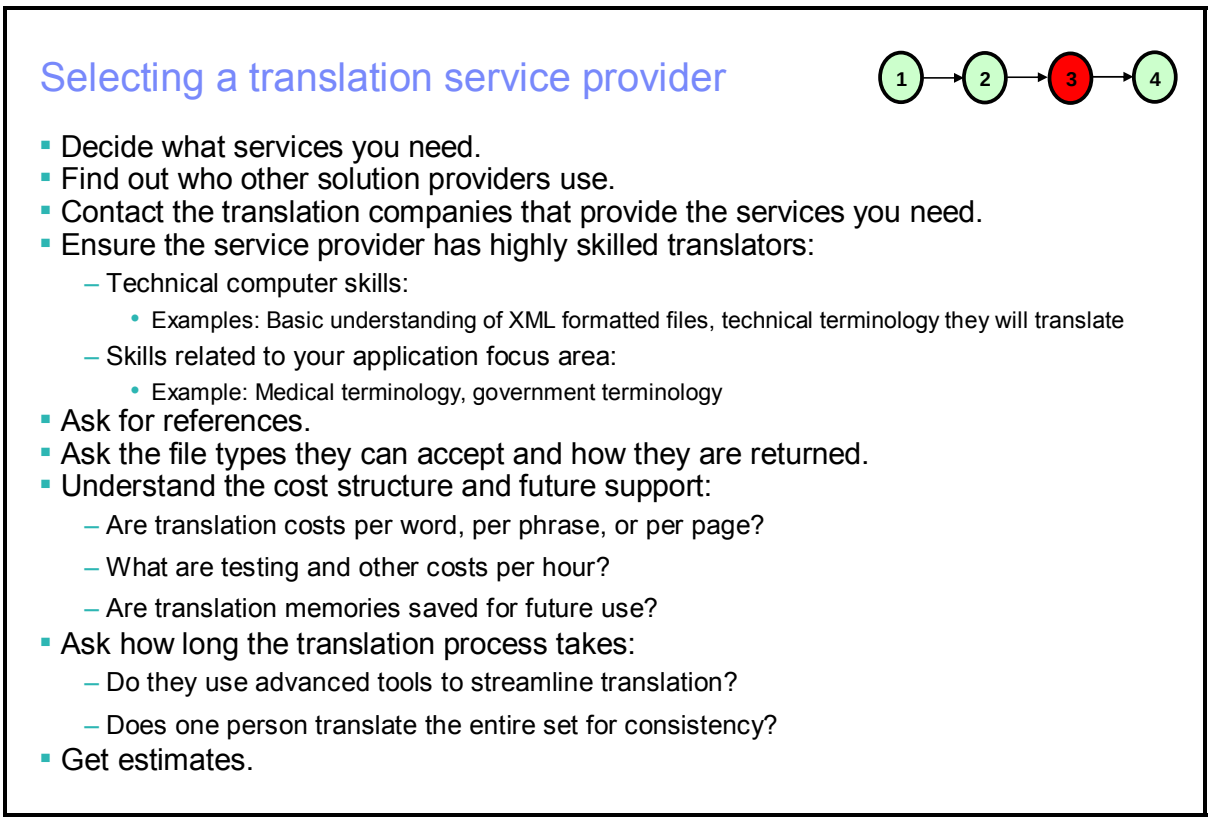
You can read about the WebSphere machine translation tool to get an idea of what a typical translation product offers by going to the Web site listed in the **Resources** section.



Manual translation

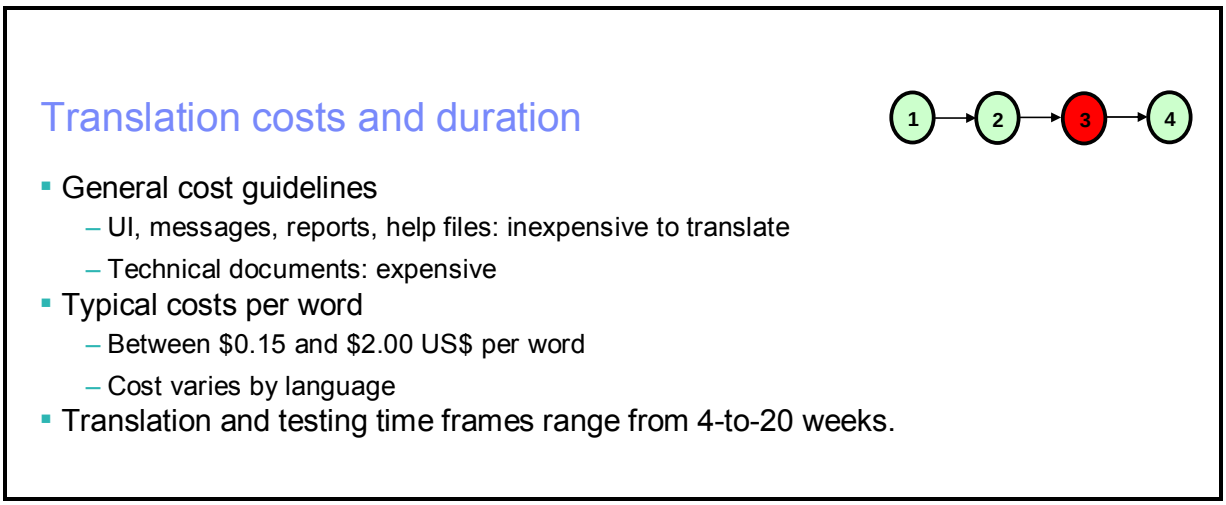
Manual translation is the best solution for translating textual data and messages. This is the method that IBM uses to translate the text for its software products.

This chart shows the typical kinds of services that translation companies offer.



Selecting a translation service provider

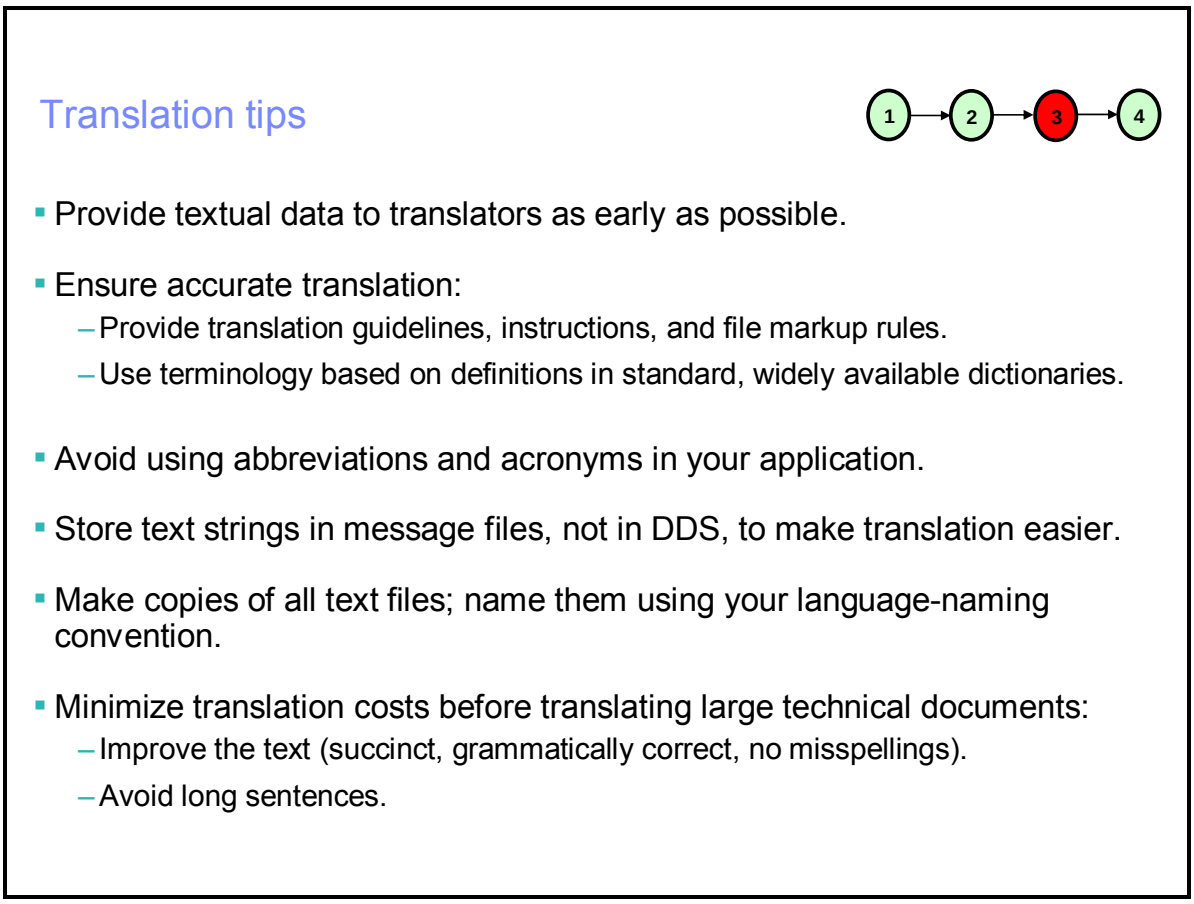
Here are tips to help you select the best translation service provider, based on your needs. It is always best to get referrals from other solution providers who have used translation services. IBM offers translation services at cost to IBM solution providers.



Translation costs and duration

The user interface is generally inexpensive to translate. Thick, technical documents cost the most. Translation companies often charge a per-word rate for translation that can range from $0.15 to $2.00 (US $) per word, depending on the language and the service provider. Other services usually have an hourly rate.

Translation and testing time frames can range anywhere fro 4 to 20 weeks.



Translation tips

Here are some translation tips to help you reduce costs and obtain results that are more efficient.

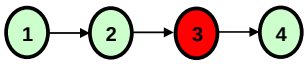
Make textual data available to translators as early as possible. Provide them with translation guidelines and instructions to ensure correct translation. You might also need to provide file markup rules if they have not worked with your types of source files.

To ensure accurate translation, use terminology that is based on definitions that can be found in standard, widely available dictionaries. Additionally, provide a glossary of nonstandard terms to the translators. Avoid using abbreviations and acronyms in your application, as they can be impossible to translate. If you must use them, define them in the glossary.

Store text strings in message files, rather than in DDS, to make translation easier. Make a copy of all text files, and name them using your language naming convention. Provide this set to the translators.

Consider improving the text to minimize translation costs before translating large technical documents. Text must be succinct, grammatically correct, and free of misspellings. Avoid long sentences; use similar terms consistently to maximize reuse. Remove sections of text that are duplicates of the help text.

Textual expansion

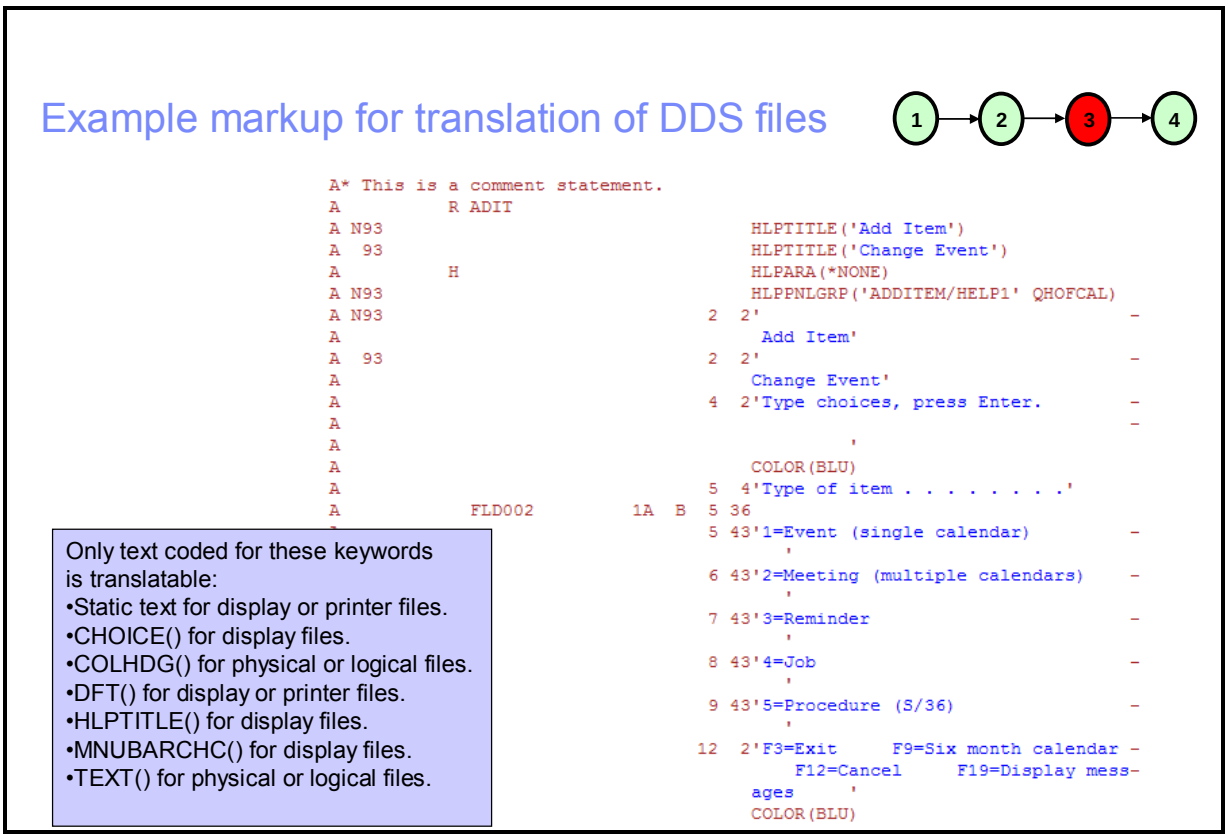


Example: The string in DDS **Part description** is 17 characters long. After translation to another language, between 14 and 17 characters of extra space are necessary for the translated string (a total of 31 to 34 characters).

Number of characters *including spaces and punctuation marks within the string	**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%
71 or more	30%

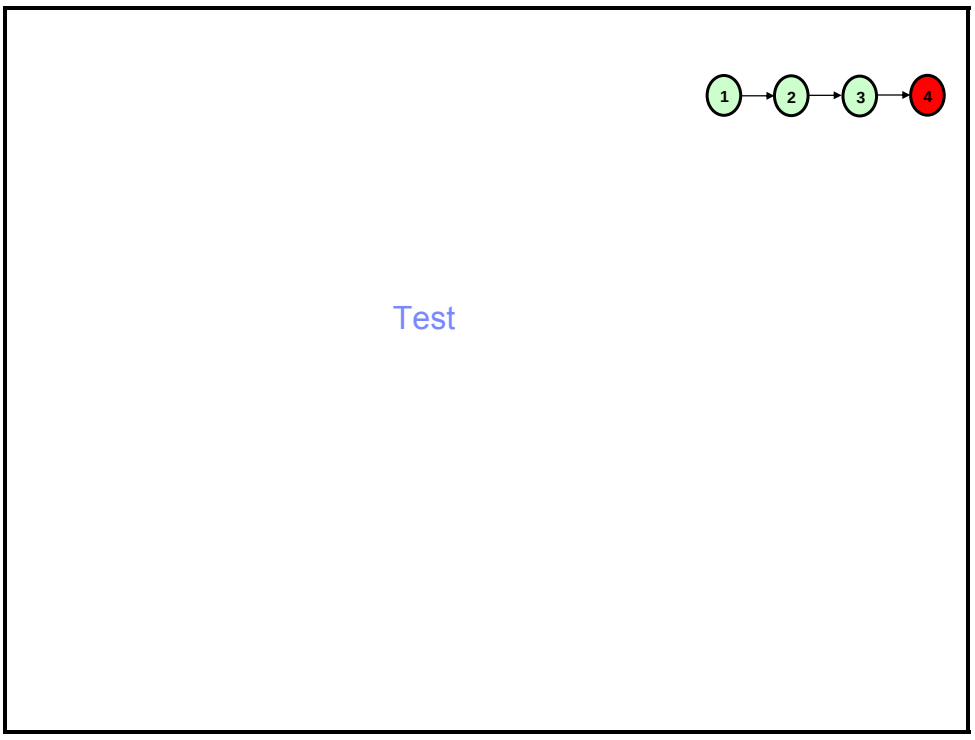
Textual expansion

You need to adjust DDS screen layouts to allow for expansion of the text volume. This chart shows some estimated expansion amounts that depend on the length of the string that requires translation.



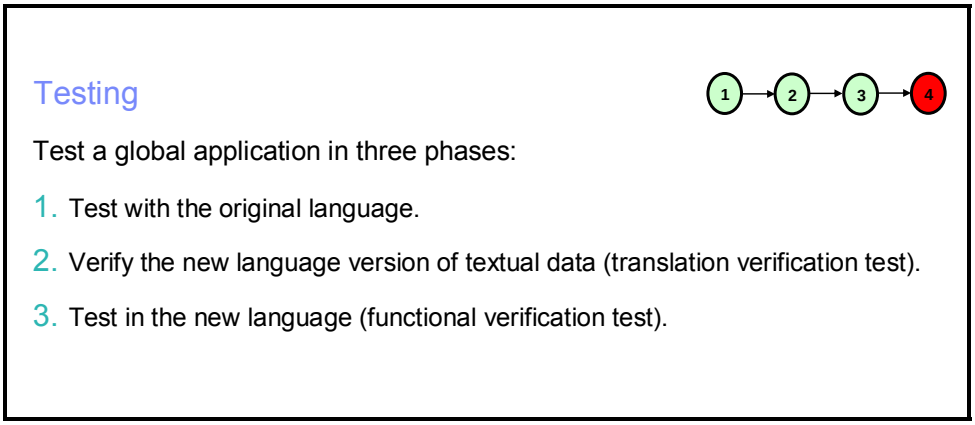
Example markup for translation of DDS files

Translators typically have tools that allow them to view a file requiring translation. Usually, the tool is designed to understand the contents of the file and to highlight which portions require translation. You can see here (in blue) which items require translation.



Test

This section discusses testing, the final important step.



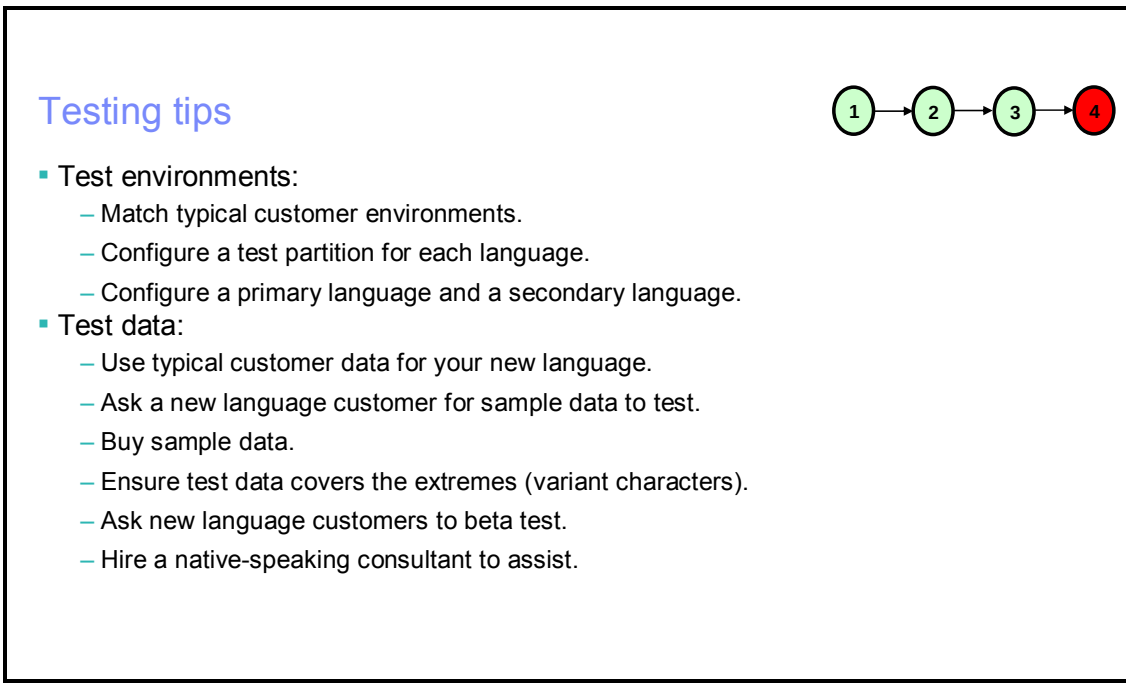
Testing

There are three beneficial phases for testing a newly global application.

Test the global application with the original language to ensure that current users will not experience problems. Ensure that culturally sensitive data still appears correctly. Additionally, you will need to process global data to make sure that it appears correctly.

You will also need to verify the new language version of textual data (translation verification test). Test the textual data using the global application. Confirm that the appropriate translation exists, appears correctly, and makes sense in the UI instance. Translators must lead this testing; programmers need to assist.

Test the global application in the new language. Be sure to verify all functions. Specifically, make sure that the application functions properly, any culturally sensitive data appears correctly, and global data is processed and appears accurately.

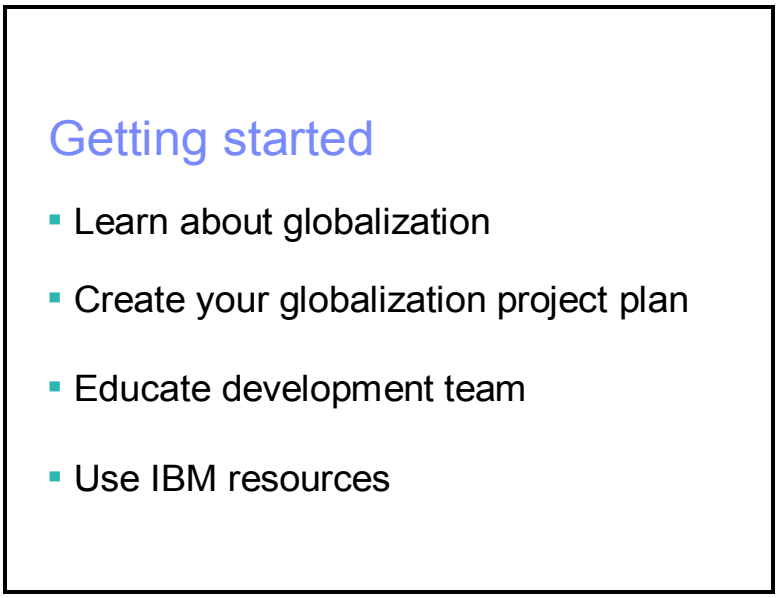


Testing tips

Here are a few testing tips to help you achieve results that are more accurate.

Use test environments that match typical customer environments. Configure a separate partition for each language version to test. Configure the system with a primary language and a secondary language. Ask new language customers to perform a beta test.

When testing data, use typical customer data for your newly supported language. Ask a new language customer for sample data to prime your test database, or buy sample data. Ensure test data covers the extremes (variant characters). Hire a native-speaking consultant to assist with testing the new language version.



Getting started

Sometimes, the biggest obstacle to accomplishing a globalization effort is figuring out where to begin. Here are some tips on getting started with a globalization project.

Learn about globalization. Review the iSeries roadmap. Take the self-study courses that IBM offers. Visit the Web sites provided in the **Resources** section of this course.

Create your globalization project plan. Analyze the language and cultural requirements of your particular users. Examine your application's database, RPG, and DDS code. Define the scope of your globalization tasks and plan how to stage the changes.

Educate your development team. Learn about Unicode support on the System i platform, and decide how to divide the various efforts of changing the code.

Use the full scope of IBM resources at your disposal. Obtain free consulting from the IBM ISV Business Strategy and Enablement group. Additionally, consider IBM Globalization Services for translation and language testing.

Summary

To stay competitive in the modern business world, globalization is no longer simply a good idea; it is a necessity. This course discussed in detail the basic concepts involved with the globalization process. You examined four major steps: analysis, implementation, translation, and testing. Example scenarios offered real-world evidence of the best approach at determining a plan of action for your globalization project. In addition, you learned about various tips and suggestions for globalization. There is also an extensive **Resources** list to explore globalization further.

Appendix A: Glossary

This appendix provides you with a glossary of terms used throughout this paper.

Character Data Representation Architecture (CDRA)	Defined by IBM to realize consistent representation, processing, and interchange of coded characters; implemented as CCSIDs in the i5/OS environment.
Character Set	A defined set of characters. Specific collections of characters represent textual information. Generally supports more than one language. Example: Latin-1 character set supports most Western European languages.
Coded Character Set Identification (CCSID)	Integrated concept of encoding scheme, character set, and code page.
Code Page	A set of assignments of characters to code points.
Code Point	A unique bit pattern that represents a character within a code page.
Culturally-sensitive	Data or information defined or represented specific to a culture.
Double Byte Character Set (DBCS)	A set of characters from languages that have more than 128 unique characters where it takes 2 bytes to store each character. Languages: Simplified Chinese, traditional Chinese, Korean, and Japanese.
Encoding Scheme	A set of rules that is substituted during character conversion for any characters in the source coding representation that do not have a match in the target coding representation. EBCDIC is the i5/OS encoding scheme. ASCII is the encoding scheme on PCs, UNIX®, and the Internet.
Endian (Big or Little)	Format for storing Unicode characters. Non-Intel machines store characters in Big Endian format. Intel machines store characters in Little Endian format. A byte order mark (BOM) indicates the format for a given set of Unicode characters.
Globalization	The proper design and execution of systems, software, services, and procedures so that one instance of software, executing on a single server or end-user machine, can process multilingual data, and present data culturally correctly in a multicultural environment.
Internationalization	The process of producing a product that is independent of language, script, culture, and coded character set. An Internationalized product is not usable unless localized to a specific region.
Internationalization Components for Unicode (ICU)	An open source development project that includes a set of APIs for processing Unicode data. ICU APIs were ported to the System i platform to support C applications. RPG code typically does not use them.
Locale	A set of values that defines a user's language, country and any special variant preferences that the user wants to see in the UI. The System i platform supports locales for C applications. RPG code typically does not use them.
Localization	The process of adapting an internationalized product to a specific language, script, culture, and coded character set. Translations are done at this point.
Machine Translation	Automatic translation of language text by computers at run time.
Multilingual	Supports simultaneous use of characters from various languages.
National Language Version (NLV)	A version of the i5/OS operating system that contains a predefined set of language-dependent values (such as: date and time format, sort sequence, and so forth). When ordering System i hardware, you specify which NLV is the primary language. You can install additional NLVs as secondary languages.
Single Byte Character Set (SBCS)	Refers to languages that have less than 128 unique characters in their scripts. Requires 1 byte to store each character.
Sort Sequence	Controls the sorting of data prior to showing it to an application user. A sort sequence can be defined using logical files or SQL views.
UCS-2	A subset of UTF-16 that does not support all defined Unicode characters. (UTF-16 also supports combined characters and surrogates.)
Unicode Standard	An industry-defined universal encoding scheme for written characters and text that enables the exchange of data internationally. It allows one field value to be multilingual.
Unicode Transformation Format (UTF)	An algorithm that maps every unique Unicode value to a unique byte sequence. You can choose from three formats: UCS-2, UTF-8, UTF-16, and UTF-32. The default on the System i platform is UTF-16 (Big Endian). RPG uses USC-2.

Appendix B: Resources

These Web sites provide references to supplement the information contained in this document.

- IBM eServer iSeries Information Center
  http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp
- IBM eServer p5 Information Center
  http://publib.boulder.ibm.com/infocenter/pseries/index.jsp
- IBM Publications Center
  http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US
- IBM Redbooks®
  http//www.ibm.com/redbooks
  - e-business Globalization Solution Design Guide: Getting Started (SG246851)

DB2 UDB for iSeries References

- IBM iSeries Information Center
  http//publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp
  Note: Click **Database**. Then, see **data types**, **SQL sort sequences**, and **CCSIDs**.
- DB2 UDB for iSeries Web site
  http//ibm.com/servers/eserver/iseries/db2/
  Note: This site includes the latest information on DB2 UDB product, tips, and concepts.

Printing references

- DDS Reference: Printer Files – Appendix B. Unicode Considerations for printer files
  http//publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp
  Note: Select **Programming** > **DDS** > **Printer files**
- AFP Architecture: Using OpenType Fonts in an AFP System (G544-5876-02):
  http://www.**ibm.com**/support/docview.wss?uid=pub1g544587602

IBM globalization

- IBM globalization Web site
  http://www.**ibm.com**/software/globalization
- iSeries globalization Web site
  http://www**.ibm.com**/servers/eserver/iseries/software/globalization

Globalization and Unicode self-study education

- Globalization: An Overview
  http://www.ibm.com/servers/enables/site/education/abstracts/88ca_abs.html
- Making Enterprise Applications Globally Available in the Modern Business Setting
  http://www.ibm.com/servers/enable/site/education/ibp/9dce/index.html
- IBM home page for International Components of Unicode (ICU)
  http://www.ibm.com/software/globalization/icu/index.jsp
- Introduction to Unicode for i5/OS on IBM eServer iSeries
  http://www.ibm.com/servers/enable/site/education/abstracts/8fce_abs.html
- Enabling Unicode in RPG Applications that Run on the IBM iSeries Systems
  http://www.ibm.com/servers/enable/site/ education/abstracts/9dc2_abs.html
- JD Edwards Unicode white paper
  http://www.ibm.com/support/techdocs/atsmastr.nsf/5cb5ed706d254a8186256c71006d2e0a/61aa3e92d01b8a6586256ebd00450415/$FILE/iSeries%20Unicode%20Upgrade%20-%202.pdf
- Unicode Web site
  http://unicode.org

IBM Globalization guidelines

- Overview
  http://www.ibm.com/software/globalization/guidelines/index.jsp
- Quick reference
  http://www.ibm.com/software/globalization/guidelines/outline.jsp

Java internationalization

- Course: Internationalization with Java: Introduction to Terminology and Architecture
  http://www.ibm.com/servers/enable/site/education/abstracts/9912_abs.html
- Sun Java Internationalization
  http://java.sun.com/j2se/corejava/intl/index.jsp
- Java Internationalization: An Overview
  http://java.sun.com/developer/technicalArticles/Intl/IntlIntro/
  Abstract: Exploring the types, structure, creation, and usage of **Java** resource bundles, this article shows you how to create a localizable program.
- Tutorial
  http://java.sun.com/docs/books/tutorial/i18n/
- Trail: Internationalization, by Dale Green
  http://java.sun.com/docs/books/tutorial/i18n/
  Abstract: These lessons show how to internationalize Java applications. Internationalized applications are easy to tailor to the customs and languages of end users around the world.
- Java Internationalization by Andrew Deitsch and David Czarnecki
  http://www.javainternationalization.com/blog/
- Java Internationalization; O'Reilly book by Andy Deitsch, David Czarnecki
  Abstract: This book shows how to write truly multilingual software, using the Unicode Standard.
- Frequently asked Q&A
  http://java.sun.com/j2se/corejava/intl/reference/faqs/index.html

Additional information

- IBM International Components for Unicode
  http://www.ibm.com/software/globalization/icu/index.jsp
- IBM WebSphere Machine Translation
  http://www.ibm.com /software/pervasive/ws_translation_server
- APPCON
  http://appcon4.com

Trademarks and special notices

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	eServer	i5/OS	DB2
ibm.com	iSeries	WebSphere	DB2 Universal Database
the IBM logo	Redbooks	System i5	iSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

All statements regarding IBM future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. Contact your local IBM office or IBM authorized reseller for the full text of the specific Statement of Direction.

Some information addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function, or delivery schedules with respect to any future products. Such commitments are only made in IBM product announcements. The information is presented here to communicate IBM's current investment and development activities as a good faith effort to help with our customers' future planning.

Photographs shown are of engineering prototypes. Changes may be incorporated in production models.