A Proposal for a Global Input Method

By Troy Korjuslommi, Oct 18, 2008 (rev 3 of 2009-06-10).

Description of the Problem

Users of keyboards around the world typically have a national keyboard which they are the most familiar with. For some languages, it is also common to use an Input Method (IM) for entering text. An IM is a piece of software which provides a service very similar to a keyboard driver, but includes some extra functionality. IMs are used especially for writing Japanese, Chinese and other such languages which have more letters than fit on a keyboard.

At least the most common personal computers such as GNU/Linux, Windows and Mac OSX support Input Methods.

I have identified six scenarios for users of keyboards, based on their needs to enter text.

Scenario 1. User is using her national keyboard and needs to enter text in her own language.
Scenario 2. User is using her national keyboard and needs to enter text in a foreign language.
Scenario 3. User is using her national keyboard and needs to enter text in multiple foreign languages.
Scenario 4. User is using a foreign keyboard and needs to enter text in her own language.
Scenario 5. User is using a foreign keyboard and needs to enter text in a foreign language.
Scenario 6. User is using a foreign keyboard and needs to enter text in multiple foreign languages.

A system devised on a national level, and used in combination with IMs for other languages, should be a suitable solution for scenarios 1-3,5,6, but not for scenario 4.

In scenario 4, the user will typically have to enter certain letters of her own alphabet frequently. The repertoire of the needed characters is probably limited to 1-5 characters (e.g. a Finnish user on a French keyboard should get by with finding the letters ä,ö and @). The user should ideally be able to enter the characters using the same method, regardless of which foreign keyboard is being used. This means that once the user has learned how to enter her own language's characters on one foreign keyboard, she can use the skill on all foreign keyboards. It is important that as few key strokes as possible are required for entering text, as the user will be needing them often.

It should be further noted that if the user is entering characters of a single language, the problem is identical to scenario 4, regardless of whether the keyboard is foreign or whether the language is her own (scenarios 2 and 5). In other words, scenario 4 can be rewritten to cover scenarios 2,4 and 5 as follows:

Revised scenario 4: User is using a keyboard and needs to enter text in a single language, other than the one the keyboard was designed for. Further, an IM for the targeted language is not available on the system.

There can be a number of reasons why a particular IM is not available. It could be that it is not installed, is not available for the platform, cannot be installed (e.g. due to lack of administrative privileges), or the national keyboard is not compatible with the IM in question (because of missing keys etc.).
The problem in the revised scenario 4 is faced by users who visit Internet cafes, or who must work in a

foreign office, or use a foreign keyboard for some other reason. The scenario also applies to multilingual users. The common factors are that these users need to produce text for some language which is not supported by the keyboard they are using, and they need to enter the characters frequently.

What would these users want? Ideally they would want to enter text just like they do on their own national keyboards. This goal is not achievable, however, because of the sheer number of keyboard layouts and the many conflicting key assignments (qwerty, azerty, languages with many composed characters etc.). Also, the extra keys they need are usually not engraved on the keyboard. Because any combination of (x) physical layout and (y) the user's desired layout is possible, it is impossible to design a universal system where users always get exactly what they want. When a user sits down in front of a computer, we know x (the physical layout), but not y (user's desired layout), and we know that we want the new layout to be as close to y as possible. Therefore, we can devise a scheme where we ask the user for the value of y (the desired layout), so that we can provide the user a layout which is optimized for x and y. Designing a a layout for each combination of x and y is not really a problem, at least for many of the combinations, since we can restrict the keys used in the layout to a set which is available on all values of x (national keyboards). A problem arises if we consider the fact that some users will also use some characters from outside y (e.g. symbols from a neighboring country), which they have available on their national keyboards at some nationally defined locations (e.g. Sami,German,Danish characters on a Finnish keyboard). In other words, users' desires differ; y is not really a known quantity.

Karl Pentzlin's design (for a revision of the ISO- IEC 9995-3) is comprehensive. It includes a very large set of characters. It is not very hard for users to learn. It also allows users to enter all characters needed in their desired layout (y), and also characters from outside y, such as symbols. However, it requires users to memorize key stroke combinations for all characters they need, or the computer's interface must contain a comprehensive set of instructions (e.g. national tables for users to find their own characters). We must also bear in mind that some keys on any keyboard are inherently more convenient to use than others. If we have predefined positions for all characters, then some keys are always less convenient for somebody.

The use of diacritics is sometimes needed, especially in some languages, but most users prefer to enter combined characters from their own language with a single key stroke.

A very common scenario is that the national keyboard (the local keyboard) has keys which the foreign user doesn't need. The keys assigned to local uses on the national keyboard could be used to implement some of the characters in the user's own set, the characters she needs the most often. E.g. a Finnish user using a Spanish keyboard could have 'ñ' mapped to 'ä' and 'ç' to 'ö'. This is only possible if we have keyboard layouts designed for each combination of x (national keyboard) and y (user's desired layout), or if the user is able to modify the mappings herself.

If we are to use an IM for entering characters, then we should take advantage of all modern facilities available to IM designers. E.g. Microsoft's Japanese IM provides various convenience facilities such as dictionary lookups, with descriptions of the word options given to the user. There is nothing which prevents a modern IM from using the network, or from presenting the user with graphic presentations which are pleasing to the eye, with menus and other facilities.
An input method for Internet cafe users needs to cater to the needs of various users from various locales. The environment is a US-ASCII or a national keyboard. Each user group has a finite set of characters which needs to be produced. In most cases with Latin characters, the number of characters is small. In some cases (Cyrillic, Greek) it is larger. In addition to the characters, some symbols must also

be produced. It would be ideal, from the user's perspective, if we could turn the keyboard to a mode where the necessary characters and symbols are produced using either using the key directly (or w/ shift), or using AltGr (or w/shift).

Which is more important for users, having the same key positions on all foreign keyboards, or having the most needed characters in easy to use key positions on the foreign keyboard? Can we provide users with both options?

How do things look like from the user's perspective? What can we reasonably expect the user to know? Very little. Since the user is foreign to the system, and will possibly only use it rarely, the number of things to memorize should be limited to the minimum.

The Proposed Solution

The above described issues were the premise for this proposal.

This proposal will address scenarios 2-6. Scenarios 3 and 6 include multiple target languages. This proposal includes a means to assist in these cases, when the number of languages is limited. However, this solution is not designed to assist users who need to enter random Unicode characters and symbols.

Below, I will describe the new system I devised as a solution to these problems. In the end you will find explanations of some of the terms I use frequently in the text.

Since there is a need to have a full IM anyway, why not come up with a method where the user can select from multiple different layouts, and select the one she likes the best? She could choose a national set of characters, or one with some of her own characters. She also should be able to specify the key positions for entering the characters.

Of course, it would be cumbersome to specify a new layout each time. So why not allow the user to save the settings in some global location? Why not have servers where the user's own settings are stored, and can be retrieved from anywhere in the world?

By allowing users to specify key positions, we can allow everybody to use the ergonomically best positions. We can also deliver a large character repertoire. Actually, the repertoire can include all characters and symbols in Unicode.

When a user arrives at a new computer, there should be a single button (or a well known command which is common to all systems) which the user can use to get started. Reasonably, this method should be a menu or a menu item. This is achievable on all personal computer systems, and even on portable devices such as phones.

Brief Description of the Solution:

Users are provided with a single menu or menu item for configuration. The configuration screen allows users to configure the system and to load a new layout. Users also select a key combination to switch between the keyboard layouts (one layout is the national keyboard layout for the physical keyboard, and the other one is the layout loaded by the user). The default layout is the one provided by the national keyboard and its extensions. The user loaded layout is superimposed on top of the national layout, so any keys not specified in the user loaded layout will work the same as they do in the national layout. Also, when switched back to the national layout, the default layout will continue working as

before.

Requirements from a Compliant System:

-No administrative privileges required to use.
-Single menu or menu item to activate configuration.

Needs from the Method:

-Must work on any national keyboard which is able to produce the ASCII letters "A-Z,.+-".
-Minimal keystrokes to produce characters.
-Must be able to produce all national characters and symbols for the languages which are to be supported.
-Must work even without a network connection.

The system is comprised of the following components within the operating system:

1. Method for activating the configuration menu. This should be a single menu or menu item.
2. Configuration system. This is the system used to select a User Customized Layout (herein abbreviated as UCL) and to save one.
3. Method for switching between keyboard layouts (local layout and the UCL).
4. Method for latching to a UCL (for entering a single character from the UCL).
5. IM which implements the UCL.

Other related components within the operating system:

1. National keyboard layout.
2. Other IMs installed on the system.

In addition, there are servers which store the UCLs. These are regular HTTP (www) servers which store the UCL files. No additional functionality is needed from the servers for providing a service whereby users can download UCLs. In order for users to be able to save new UCLs on the servers, some extra functionality is needed. Since the UCLs are just text files, and are generated on the user's computer, implementing a service for storing UCLs will be simple. A protocol for the saving process related communications between an IM and a server is described below.

The topics below have more detailed descriptions:

A. Configuration System
B. UCL Design Tool
C. Switching/Latching to a UCL
D. UCL Format
E. Protocol for Reading a UCL from a Server
F. Protocol for Saving a UCL to a Server
G. Additional Notes
H. Terminology

A. Configuration System:

This is the system used to select a UCL and to save one.
The user is presented with a screen where she can select from a number of choices:
(a) Create a new UCL. This is a tool which allows the user to select Unicode characters from tables. A search facility is used to locate characters.
(b) Download a UCL from a server.
(c) Select a UCL from the cache.
(d) Load a UCL from a file.
(e) Save the UCL to a file.
(f) Save the UCL to a server.
(g) Configure the commands to switch/latch between the default layout and the UCL.

The IM should have a command for saving the UCL to a file. It should also give the user the ability to load a UCL from a file. Thus, the user could save the file for future editing before storing it to a server. Once she was happy with it, she could save it to a server. The file should have the suffix ".html", so it could also be viewed in a web browser.

A computer should cache some common UCLs, such as those created for various locales, and also at least twenty most recently used UCLs. The user should be asked whether to retain a UCL in cache after she logs out. Thus, users who frequent a particular computer could have their UCLs cached. Caching will make loading the UCL faster, and will also enable use of the UCL when the computer is not connected to the Internet. If no UCLs are found and no network connection can be established, the user can always create a UCL manually.

For phones and other systems with a limited GUI, there could be a universal (or a device type specific) command for loading a new UCL. The command would prompt for the hostname and UCL name, and then load the UCL. A text based interface (curses etc.) could also be used on such systems.

An important criteria for this IM is that it should require no administrative privileges to use. It must work on any national keyboard which is able to produce the letters needed to enter the hostname and UCL name, so that a UCL can be loaded to the system.

The user should be given recommendations on selecting names. E.g. A Finnish user designing a private Finnish keyboard could select "fi" as the name for a generic keyboard UCL. For Spanish keyboards, she could create another UCL called "fi/es". Or she could name it "suomi/espanja," if she so preferred.

B. UCL Design Tool :

The UCL design tool allows the user to create a customized UCL.

The UCL design tool should have a map with all unicode characters laid out in tables, and a search facility for looking up characters by name, description, code value and code value range. The user should be shown the visual representation of the characters, as well information about them, so she doesn't accidentally mistake one character for another. The information should include the codepoint value, notes on the countries where the character is used, the full name of the character, and other information which can help the user.

There should also be tables with some preselected sets for locales found in the CLDR. The exemplar characters from CLDR could be the basis for such tables. By selecting multiple languages, the user could easily select characters from them all.

When the user creates an entry for a lower case character which has an upper case counterpart, the tool should, by default, also generate an entry for the capital letter, using shift and the same key position. The tool could ask for confirmation from the user before actually adding the entry.

One option is to implement some default mappings, such as mapping the keys “0-9,.-” to various diacritic characters. This concept, which was specified in Karl's proposal, can be an extremely useful one, since having one easily describable location for diacritics would make their use much easier. Currently people have to learn the names of diacritics (diaeresis etc.), which are not widely known even among the people who have them on their national keyboards. However, diacritics should be rarely needed, as most users have only a limited set of (combining) characters they need to produce. Therefore, they can specify a unique key for each character. Of course, in some locales (e.g. Lithuanian), diacritics are used so widely that they must always be present.

Automatically Created UCLs:

It should be possible to create default variants for all the major locales using CLDR's exemplar characters. This way we would have a sample UCL for each locale in CLDR. User groups could modify these UCLs, or create new ones. We could even call these default groups the CLDR default sets. Symbols used in each locale should be added to these sets (the CLDR has some symbols, but there is no comprehensive listings, such as the exemplars, which could be used as-is). The key positions could be selected using some simple algorithm. E.g. create a list of preferred locations from an ergonomic perspective, and then add the characters there in a random order.

Using this method, we are also in the position to allow users to specify new positions for characters such "$@{}[]()", as it suits them. These and other symbols are often in different locations on national keyboards, and their use varies by user group. Programmers need them often, and having a programmer specific UCL, programmers could be assured that they can find all special symbols which are found on a US-ASCII keyboard. The same applies to mathematicians. Users of mathematical symbols could define one or more UCLs which included all characters they need.

There should be recommendations as to which keys should be preferred, to help users and designers of UCL to implement easy to use designs. Left handed and right handed users, for example, might have different needs.

C. Switching/Latching to a UCL.

Switching:

A method for switching between keyboard layouts (local layout and the UCL).
(a) One command (e.g. CTRL+'.'). Every time the command is executed, the system switches between the default layout and the user's chosen UCL.
(b) Two commands (e.g. CTRL+'+' and CTRL+'-'). Each command always switches to its designated layout. E.g. CTRL+'+' could always switch to the default layout, while CTRL+'-' could always switch to the user's UCL.

Latching:

Method for latching to a UCL.
A command (e.g. CTRL+',') which allows the user to enter a single character using the UCL, and then changes back to the previous layout. (Latching is desirable, but optional, as the technology is not currently in use. It is uncertain how to best implement this functionality. It is probably not a difficult issue to solve, however.)

D. UCL Format

There should be a well defined, operating system independent, protocol for storing UCL files. As for character encoding, UTF-8 is suitable for this purpose.

We can use key/value pairs (records) to describe a mapping between a key on the keyboard and the output character. Records are delimited by newlines. The set of records is contained within a single XML element, which itself is contained within a valid XML document.

Each record contains a key/value pair, delimited by ':', a colon. The left hand side of the pair, the key, is the key to enter (e.g. "a"), plus any modifiers (shift, altgr etc.). The right hand side, the value, is one or more Unicode characters (e.g. "U+00AB U+00BB") to be produced.

The Key:

The goal is to make things work predictably for a user who relies on keyboard engravings. Therefore, if the user's UCL uses the 'z' for one of the positions, she should be able to press the key engraved with 'z' on qwerty and azerty keyboards, and get the same character as output. We also want shift+key (level 2) to produce the upper case variant for those characters which have lower/upper case variants. Similarly, we want the user to be able to map a key to levels 3 and 4 (AltGr and shift+AltGr).

Keyboard layouts are usually described as mappings from a keycode (generated by the keyboard) and modifier (shift, ctrl, alt, altgr) combination to an output character. The keycode which gets generated by a keyboard is the same, regardless of the modifiers. I.e. when entering 'a' and 'a' + shift (i.e. 'A'), both produce the same keycode, with 'A' having the SHIFT modifier also active. The keycode which corresponds to 'ä' on a Finnish keyboard corresponds to “'” (single quote) on a US-ASCII keyboard.

When designing a system where settings are overlays which are used with multiple keyboard layouts, we cannot use keycodes directly. The reason for this is that keycodes depend on the position of the key on the keyboard, so that when comparing a qwerty and azerty keyboard, the 'q' key on the qwerty keyboard generates the same keycode as the 'a' key on the azerty keyboard. These types of conflicts between keyboard layouts mean that users would have to memorize key positions on the keyboard, instead of relying on keyboard engravings. This type of IM for general use should be designed for

people who are not touch typists, and therefore rely on keyboard engravings. After all, a touch typist could just switch to her national layout on any keyboard. Since she doesn't need the engravings, she can use any keyboard the same, regardless of the engravings.

As a solution, the IM must check what would be the character generated by a keyboard to a certain key being pressed. This should correspond to the key engraved on the keyboard. This way the IM maps a keycode to a certain character. Once the IM has the keycode, it can create level 2-4 mappings for the key as well. In other words, we use the character to map to a keycode and then only check the keycode+modifier for further input. This way the user will get to use the key with the same engraving on all keyboards, even if their positions moved. This makes life a little more difficult for IM implementors, but since they have to implement a superimposed layout of the national layout anyway, the added burden on them doesn't seem excessive. Also, any design for users should focus on user needs, not on making life easy for implementors.

When the user is designing a UCL manually with the design tool, she is asked to press a key on the keyboard, and then map it to a character she wishes to produce. Therefore, the key press is received by the IM software, translated to the national keyboard's output character, and then stored as the key (the left hand side entry) of a UCL record.

For widely deployed UCLs, only the ASCII characters "a-zA-Z0-9,.-" should be used on the left hand side. In addition, the use of characters other than ASCII "a-z" should be discouraged, as "0-9,.-" could have default mappings to diacritics (or virtual diacritics), and "a-z" + SHIFT should be preferred to using "A-Z."

We allow CTRL, in addition to the other modifiers, so that users have greater flexibility. A certain key+modifier could be needed by an application, or blocked by an operating system/windowing system, so having extra flexibility can be helpful. The use of CTRL in designing UCLs should be discouraged, however.

The Output Characters:

We allow multiple Unicode characters on the right hand side. The reason for this is to allow multiple Unicode characters to be generated with one key command. This way it will be easier to generate characters which are not composable, such as some Lithuanian characters. Adding such flexibility will also allow users to invent new uses for the IM, such as entering their signature, or other frequently used data.

For security reasons, it is important that only characters which have a reasonable use in text editing should be output, and thus be allowed on the right hand side. An IM which receives a UCL with illegal values on the right hand side should strip the characters and alert the user.

The Xml Document:

A document starts with an XML declaration, and is followed by a top level element named “pre.” Inside the “pre” element we have the records. Anything after ';' is a comment until the next newline. The advantage of using "<pre>" is that the record can be viewed in any web browser. The format of the actual entries is as simple as possible, to make it easy to write parsers. The reason to allow comments is so that when a user views the document in a web browser, she can see the Unicode characters which get produced by each record. Note that all Unicode characters, including newlines, are allowed on the right hand side. The only exception are control characters and values which are not valid Unicode

characters. This issue requires some further thought, however, to ensure that no other Unicode characters have security implications. The data in a UCL should be vigorously checked by an IM, as it can potentially contain hostile data. We choose to use line delimited records, instead of XML elements, so the IM doesn't have to parse the file as XML. This means we don't have to be concerned with malicious XML documents (e.g. memory exhaustion, etc. attacks).

A sample record would read as follows:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<pre>
ñ        : U+00E4                    ; ä
ñ SHIFT   : U+00C4                    ; Ä
b ALTGR   : U+0024 U+0031 U+0030 U+0030   ; $100
</pre>
```

Or in augmented BNF:

```
record  := 1*key *(“SHIFT” / “CTRL” / “ALTGR” / “ALT”)  “:” code *(space / code) *comment
code    := “U+” 2*<ASCII CHARS IN THE RANGE [0-9a-fA-F]>
comment := “;” *uchar
uchar   := <ANY UNICODE CHARACTER EXCEPT CR,LF AND CONTROL CHARACTERS>
key     := <ANY CHARACTER WHICH CAN BE GENERATED BY A KEYBOARD EVENT>
ws      := “\r” / “\n” / space
space   := “ “ / “\t”
```

If a Finnish user used the above UCL on a Spanish keyboard, she would press the latch key, and then the key for letter 'ñ' in order to produce the character 'ä' (U+00E4). The latch key, followed by the key for 'ñ', with shift down, would produce the 'Ä' character. Using switching, the user would switch to the UCL layout, and then just press the key for 'ñ' when she needed 'ä' and 'ñ' + shift when she needed 'Ä'. Since the UCL layout is superimposed on the national layout, she wouldn't have to switch back, unless she needed the 'ñ' character from the Spanish layout. If she pressed AltGr and 'b', the string “$100” would be output.

E. Protocol for Reading a UCL from a Server

A server stores the UCL files. A user can specify a hostname (server address) and the UCL's name to download a specific UCL. Read access cannot be restricted, as any IM should be able to read the UCL, given a hostname and the UCL name. The UCL is stored as a UTF-8 encoded text file and can be retrieved using either the HTTP or the HTTPS protocol. The IM should always try using the HTTPS protocol first, and if there was a permanent error, try the HTTP protocol.

Examples:

Host: ucl.tksoft.com
Name: fi
-> URI: https://ucl.tksoft.com/fi/
and URI: http://ucl.tksoft.com/fi/

Host: ucl.tksoft.com
Name: ///de/
-> URI: https://ucl.tksoft.com/de/
and URI: http://ucl.tksoft.com/de/

Host: ucl.tksoft.com
Name: cgi-bin/fi
-> URI: https://ucl.tksoft.com/cgi-bin/fi/
and URI: http://ucl.tksoft.com/cgi-bin/fi/

The hostname must be a valid hostname and the name must be a valid file path part of a URI (URL), with the exception that the only valid characters are "a-z0-9/+-". In particular the characters "A-Z&%;?=" are not allowed. The slash is allowed so URIs can point to subdirectories on a server. The "+" characters enables spaces in file names. Multiple slashes are converted to a single slash in the URI. A trailing slash is always added to a URI. Other allowed characters have no special meaning. The strict limit on characters in names is needed so that the names can be entered from any national keyboard with ease. Names could be simple names such as "fi" or more descriptive, such as "mathsymbols." Names have no maximum length. Any possible length limits are imposed by the server where the UCL is stored.

UCL names are case insensitive. Therefore, the names “fi”, “FI” and “Fi” are equivalent. However, the actual name as it occurs in the URI is always in lower case. Therefore, if the user enters a UCL name “FI” and hostname "ucl.tksoft.com," the queries to the server will be “https://ucl.tksoft.com/fi/” and “http://ucl.tksoft.com/fi/”.

If the server responds with an error code, the IM should act accordingly, depending on the error code. Normal logic in HTTP communications should be followed. E.g. an error indicating that the resource has moved should be followed by the IM making a new request to the new address. The user should also be notified that the address has moved. If an error code indicates a failure, the user should be alerted with the message from the server. When using https, SSL certificates should be checked, and the user alerted if the certificate cannot be verified.

F. Protocol for Saving a UCL to a Server

Each server can implement its own policies, but it is recommended that the first user to register a particular name becomes the "owner" of the UCL. A server administrator can specify that only an authenticated user can create a new entry. This means that the server administrator has some control over who uses the server. This is not required, however. If the server administrator wants to allow registering new names for anyone, that is alright. If users are required to register, a policy should be in place which only allows the owner of the name (the person who created it) to create names underneath it. E.g. the owner of "ucl.tksoft.com/fi" should be the only one allowed to create an entry "ucl.tksoft.com/fi/es". This rule is recommended so that users can "take ownership"of their own layouts. This means they can create versions of their own layouts, and always use a similar name for their UCLs.

A UCL saved to a server is always new. Once a UCL is created, it cannot be edited. This means that a user can rely on any UCL to always be the same, even if it belongs to someone else. If a user wants to create a new version of a UCL, it can be saved as {hostname}/{oldname}/{newname}, so that the new name then becomes {oldname}/{newname}. E.g. "fi" becomes "fi/es".

Authentication cannot be based on username/password, since there are other methods such as HTTP authentication, authentication via a secure server, single login systems etc., and maximum flexibility should be given to server administrators.

It is recommended that the IM and servers implement the following protocol for saving a UCL:

The user clicks a "save to server" button in the IM, which prompts the IM to connect to the server and submit the UCL to the server, using a randomly generated token (the token must contain only the characters "a-zA-Z0-9" and be 64 characters or less in length). The submission is made using a POST request. The server accepts the request and saves the UCL in a file, associated with the token. The server then returns a 200 OK response. The POST request URI is PROTO://{hostname}/create-ucl/TOKEN, where PROTO is either https or http, whichever one succeeds (with https tried first), and TOKEN is the token generated by the IM. (If the server doesn't allow the request, it should return an error message, such as “501 NOT IMPLEMENTED” or “503 SERVICE UNAVAILABLE.” These are only suggestions. Normal HTTP protocol rules apply.) The IM then launches a web browser, using the URI PROTO://{hostname}/save-ucl/TOKEN. Once the browser is launched, the client can proceed to authenticate with the server. The server now has the UCL stored temporarily, and the TOKEN to identify the UCL is sent in the request. After authentication, the user must still select a name for the UCL. Once all this is done, the server has all the information it needs to save the UCL permanently. From a user's perspective, only three steps are required: (1) click the save button, (2) login to the server using a web browser, and (3) name the UCL.

The protocol described above for saving a UCL is only the default way to add a UCL. An implementation could choose to reject updates from IMs, use a web form with authentication, email, WAP or any other method. A particular server could even deny all editing access.

It is imperative that it is convenient for users to upload a UCL to a server. Therefore, servers can also implement other methods for saving a UCL. One solution would be for the server to have a form which allows the data to be uploaded through a file upload facility, whereby the user selects a file to be uploaded (which she has saved to disk previously). This way a user can save a UCL which she has edited manually, e.g. to add comments.

G. Additional Notes:

The reason we must have key commands for switching and latching is so that users can enable the IM in a fast and easy manner. The reason the commands must be customizable, is because if we had global keys for switching and latching, it would be difficult to have assurances that no existing or future application used the same combination for some other purpose. Also, it should be as easy as possible for users to switch between operating systems.

It would be ideal if the user was able to designate more than one switching/latching command, so that there would be more than one UCL available to the user.

All keys not specified in a UCL retain their functionality on the national keyboard.

A protocol extension could be added to this protocol, whereby a UCL can include a list of national keyboard layouts for which it is suitable. An IM could then query a server, and give the user options for UCLs to download, with the most suitable layouts listed first.

Users who need to enter random Unicode characters can use another Input Method or technology to enter those characters. If it is deemed necessary, such a method could also be made to work in conjunction with this IM.

H. Terminology:

-UCL: User Customized Layout
-Swithing means that the layout becomes the active layout for all future input.
-Latching means that the layout becomes the active layout only until a single a character is output. The previous layout then becomes active again.
-IM: Input method
-Virtual diacritic: characters such as '/' as a combining character, which do not exist in unicode, but which can be used to emulate diacritic behavior by generating characters such as 'ø' by combining the characters 'o' and '/'.
-CLDR: http://www.unicode.org/CLDR/