

Solaris InputMethod implementation

06/29/2007

Index

Preface.....	1
Overview.....	1
XIM implementation.....	3
How it works.....	3
XOpenIM.....	3
XGetIMValues.....	9
XSetIMValues.....	11
XCloseIM.....	12
XCreateIC.....	12
XSetICFocus.....	17
XFilterEvent.....	18
Xmb/wcLookupString.....	19
Xmb/wcResetIC.....	19
XGetICValues.....	20
XSetICValues.....	20
XUnsetICFocus.....	20
XDestroyIC.....	20
XIMOfIC.....	20
GTK-IM implementation.....	21
Low level libraries implementation.....	21
Server module implementation.....	21
IIIM Utility implementation.....	21

Preface

This document tries to explain InputMethod implementation of Solaris 10 8/07 or later includes Nevada, Solaris Express (call them just Solaris in the later part of this document) for the maintenance engineers of InputMethod functionality on Solaris. It's not for InputMethod user or application programmer, although it may be helpful for QA engineer or user support engineer to analyze/report the issues they see or are reported by customers. This is not a specification in any means, and the contents may be modified without notice.

Overview

When examine InputMethod related issues on Solaris, there are some prerequisite knowledge regarding its implementation.

There are two kinds of applications which use InputMethod on Solaris environment. One is the application which uses XIM API, and the other one is the application which uses GTK-IM API. Each type of application uses different InputMethod modules, so investigating or fixing problem

regarding InputMethod starts with figuring out what type of application user is using. The former (XIM) type of application includes CDE/Motif applications, Java applications, Flash applications and the Xlib applications which uses Xlib/XIM APIs directly. The later (GTK-IM) type of application is all of GNOME/GTK applications includes Firefox/Thunderbird. Regarding StarOffice/StarSuite/OpenOffice.org (call them just OpenOffice.org in the later part of this document), it uses XIM or GTK-IM depends on runtime environment. If it runs on JDS session, it uses GTK-IM and when it runs on CDE, then it uses XIM.

Both of XIM and GTK-IM has the capability of dynamically loading InputMethod modules depends on user's locale or other preference, but the module switching in GTK-IM is higher level layer than XIM. It means if user preferred to use XIM on GTK applications, then GTK-IM can load XIM module to do it. In that case, all types of applications are XIM clients. Fig-1 depicts this kind of module switching briefly. These mechanisms will be explained in the following sections.

Another point which needs to be clear before looking into InputMethod related problems is the locale in which the users run their applications. The locale also determines which InputMethod modules are involved, so figuring out user's locale or whether the same problem happens with other locales are another important points for investigating InputMethod related problems.

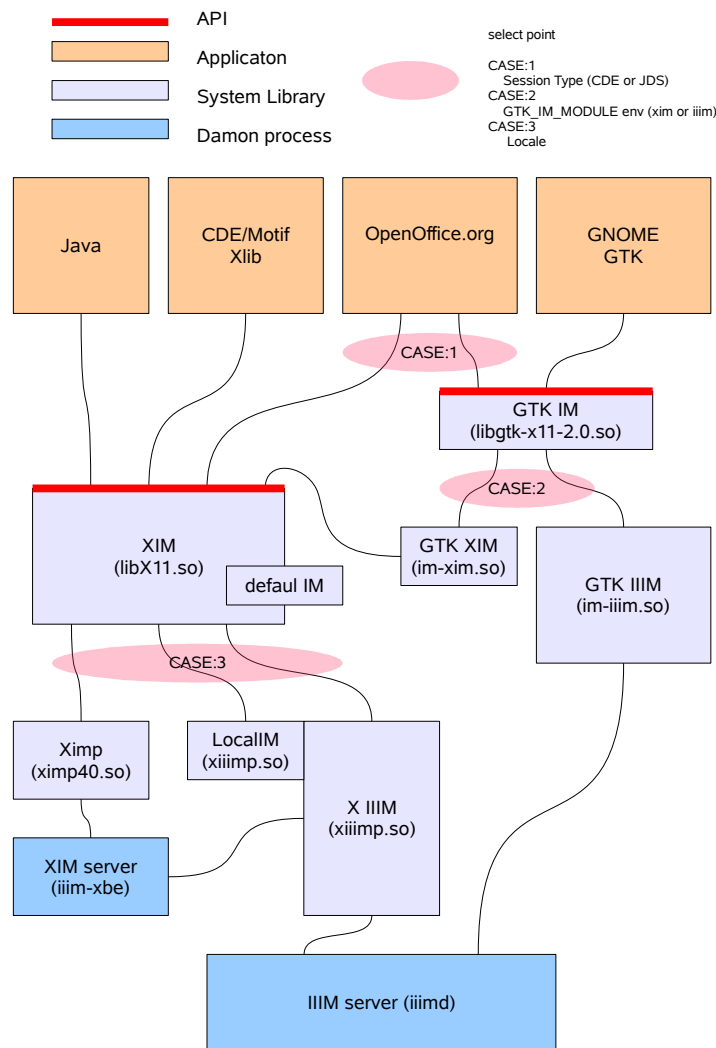


Fig-1

The following sections explain modules for XIM implementation, GTK-IM implementation, InputMethod server, and InputMethod related user program called `iiim-panel` and `iiim-properties`.

XIM implementation

This section explains the InputMethod implementation for the application/toolkit which use XIM API as the way to use InputMethod.

There are some key XIM public Xlib APIs which communicate InputMethod service. Here is the list of XIM APIs and their brief explanations.

XOpenIM	- open the connection to input method (IM)
XGetIMValues	- get input method attribute values
XSetIMValues	- set input method attribute values (internal use only)
XCloseIM	- close the connection to input method
XCreateIC	- create input context (IC)
XFilterEvent	- pass X event to input method
Xmb/wcLookupString	- obtain committed char/string from IM
Xmb/wcResetIC	- reset of the status of IC
XGetICValues	- get input context values
XSetICValues	- set input context values
XSetICFocus	- notify IC to get focus
XUnsetICFocus	- notify IC to lost focus
XDestroyIC	- destroy input context
XIMOfIC	- return IM which IC belongs to

How it works

The XIM works as follows basically.

XOpenIM create locale dependent XIM object which contains **XCreateIC** function body.

XCreateIC create XIC object which contains other XIM API bodies (except **XFilterEvent**).

XSetICFocus (or **XCreateIC** in some cases) register keyevent filter which pass keyevent to InputMethod processing object.

XFilterEvent calls registered keyevent filter and called InputMethod processing object react to keyevent. That reaction causes some callback invocation.

Xmb/wcLookupString get result string. The below is the detailed explanation for such API internal.

XOpenIM

Arguments:

```
Display *display;  
XrmDatabase *rm;  
char *res_name;  
char *res_class;
```

Returns:

XIM handle (opaque structure pointer)

Description:

This API is the trigger for InputMethod module initialization for XIM and returns the handle for XIM object. When the application calls this function, then necessary initialization includes loading dynamic modules and connect to InputMethod server depends on user's locale is done.

Source file:

`<libX11.so - IMWrap.c>`

Internal detail:

First step: Select **XOpenIM** body function depends on user's locale

It calls `_XOpenLC` `<libX11.so - lcWrap.c>` which initialize locale dependent objects include XIM object. More precisely, `_XOpenLC` loads locale dependent XLC module by using `_XsunOsDynamicLoad` `<libX11 - XsunDL.c>` and calls it's entry function, then the XLC entry function sets locale dependent **XOpenIM** body function pointer to internal structure (XLCdMethods of XLCd structure `<libX11.so - Xlcinit.h>`).

XLC entries are defined in `/usr/openwin/lib/locale/<locale name>/XI18N_OBJS` plain text file. The examples are below.

UTF-8 locales

```
XLC      common/xlcUTF-8      _XlcUnicodeLoader      # XLC_open
```

Other non UTF-8 locales

```
XLC      common/xlibi18n      _XlcGenericLoader      # XLC_open
```

If there is no `XI18N_OBJS` file under `<local name>` directory, then `_XlcDefaultLoader` `<libX11 - lcDefConv.c>` is used for loading XLC module. But in case of locale is iso8859-1 compatible like `fr`, `de`, the locale name is converted to iso8859-1 using `locale.alias/locale.dir` file. At the result, all of non UTF-8/C locale use `_XlcGenericLoader`, and only C locale uses `_XlcDefaultLoader`.

Actually both of `_XlcUnicodeLoader` and `_XlcGenericLoader` set `_XsunOpenIM` `<libX11 - XsunDL.c>` function pointer as **XOpenIM** body. And `_XlcDefaultLoader` set `_XDefaultOpenIM` `<libX11 - XSunIMIF.c>` as **XOpenIM** body. The `_XDefaultOpenIM` is the actual body of **XOpenIM**, but the `_XsunOpenIM` delegate further by calling locale dependent **XOpenIM** implementation which is defined `XI18N_OBJS` file as XIM entries. The XIM entries examples are below.

Asian UTF-8 locales (zh_CN.UTF-8, ja_JP.UTF-8, ko_KR.UTF_8 etc...)

```
XIM      common/xiiimp      _SwitchOpenIM          # XIM_open
XIM      common/ximp40      _Ximp_OpenIM           # XIM_open
...
```

EMEA UTF-8 locales (fr_FR.UTF_8, de_DE.UTF-8 etc...)

```
XIM      common/xiiimp      _SwitchOpenIM          # XIM_open
XIM      common/xiiimp      _XimpLocalOpenIM       # XIM_open
...
```

Asian non UTF-8 locales (ja, ko, th_TH, zh, zh_TW etc...)

```
XIM      common/ximp40      _Ximp_OpenIM           # XIM_open
```

EMEA non UTF-8 locales (ar, he, fr, de etc...)

```
XIM common/ximlocal _XimpLocalOpenIM # XIM_open
# ximlocal is symbolic link to xiiimp.so
```

There are several XIM lines in some locales, but only the first line is effective in most cases. Other entries are fallback for the case of pervious entry is failed, but such case must be problem to be investigated.

So there are four **XOpenIM** implementations which are used in Solaris.

_SwitchOpenIM in xiiimp.so for UTF-8 locales, **_Ximp_OpenIM** in ximp40.so for Asian non-UTF-8 locales, **_XimpLocalOpenIM** in xiiimp.so for EMEA non-UTF-8 locales and **_XDefaultOpenIM** in libX11 for C locale.

Second step: Call **XOpenIM** body function

_SwitchOpenIM

Description:

This opens IIIM InputMethod which enables selecting input language dynamically and enable multilingual input. This is **XOpenIM** for UTF-8 locales.

Source file:

[*<xiiimp.so – switchIM.c>*](#)

Internal detail:

It allocates struct `XimCommonRec` [*<xiiimp.so – commonIM.h>*](#) which can be used as `XIM/struct _XIM` [*<libX11 – Xlcint.h>*](#), as the first two members are the same.

```
typedef struct _XIM {
    XIMMethods      methods;          /* method list of this IM */
    XIMCoreRec      core;             /* data of this IM */
} XIMRec;

typedef struct _XimCommon {
    XIMMethods      methods;
    XIMCoreRec      core;
    XIMXimp         ximp_impact;      /* has to be just after XIMCoreRec */

    XlcConv         mtow_conv;
    XlcConv         wtom_conv;

    XIMPopup        popup_impact;
    XIMGUI          gui_impact;
    XIMDL           dl_impact;

    Bool            isUnicode;        /* always use UTF16 encoding */
    XIMUnicodeCharacterSubsets *unicode_char_subsets;

    /* XIM private part */
    XIMComposeIM    local_impact;     /* must be null when IIIMP only */
}
```

```

XIMIIimpIM    iiimp_impact;    /* must be null when localIM only */

SwitcherInfo *switcher_info;
} XimCommonRec;

```

The one of main roles for **XOpenIM** is to set `XIMMethod` [<libX11.so - Xlcint.h>](#) members for later use with other XIM API.

```

typedef struct {
    Status (*close)(XIM);
    char* (*set_values)(XIM, XIMArg*);
    char* (*get_values)(XIM, XIMArg* );
    XIC (*create_ic)(XIM, XIMArg*);
    int (*ctstombs)(XIM, char*, int, char*, int, Status *);
    int (*ctstowcs)(XIM, char*, int, wchar_t*, int, Status *);
} XIMMethodsRec, *XIMMethods;

```

_SwitchOpenIM set them as the following.

```

close = SwitchCloseIM;           <xiiimp.so - switchIM.c>
set_values = IIIMP_SetIMValues;  <xiiimp.so - iiimpIM.c>
get_values = IIIMP_GetIMValues;  <xiiimp.so - iiimpIM.c>
create_ic = SwitchCreateIC;      <xiiimp.so - switchIM.c>
ctstombs = _Ximp_ctstombs;       <xiiimp.so - XimpConv.c>
ctstowcs = _Ximp_ctstowcs; <xiiimp.so - XimpConv.c>

```

SwitchCloseIM, **SwitchCreateIC**, **IIIMP_SetIMValues** and **IIIMP_GetIMValues** are explained in corresponding XIM API sections.

It also calls **CommonOpenIM** [<xiiimp.so - commonIM.c>](#), **COMPOSE_OpenIM_SWITCH** [<xiiimp.so - composeIM.c>](#) and **IIIMP_OpenIM_SWITCH** [<xiiimp.so - iiimpIM.c>](#) in this order.

The tasks of the first called **CommonOpenIM** are set encoding converters (`mtow_conv`, `wtom_conv`), `core` (`XIMCoreRec`) members, `ximp_impact` member with **Ximp_OpenIM** [<xiiimp.so - ximp40.c>](#) and `dl_impact` member with **OpenDynamicObject** [<xiiimp.so - iiimpDL.c>](#). This **Ximp_OpenIM** is dummy **Ximp_OpenIM** which is different with one in `ximp40.so`. It set only string encoding conversion functions in `ximp_impact`.

OpenDynamicObject loads locale dependent dynamic loadable module if any. Current Solaris implementation does not use this.

The next called **COMPOSE_OpenIM_SWITCH** initialize `local_impact` member (`XIMComposeIMRec`) with `/usr/openwin/lib/locale/<locale name>/Compose` file.

```

typedef struct _XIMComposeIMRec {
    XIC          current_ic;
    LocalIMState *top_state;
    Bool         use_binary_table; /* True if use binary table */
    void         *map_addr;        /* for binary table */
    size_t       map_len;          /* for binary table */
}

```

```

CARD32      *state_addr;    /* for binary table */
CARD32      *def_addr;     /* for binary table */
CARD8       *str_addr;     /* for binary table */

XIMMethods  switch_methods;
} XIMComposeIMRec;

```

The last called **IIIMP_OpenIM_SWITCH** initialize iiim low level library (libiiimcf.so) for connecting iiim server module and initialize `iiimp_impact` (`XIMIImpIMRec`) member with **IMCreateHandle** `<xiiimp.so – iiimcfFun.c>`. **IMCreateHandle** connects to IM server (iiimd).

```

typedef struct _XIMIImpIMRec {
    IIIMCF_handle handle;
    Window        cb_window;
    Bool          inited;

    int on_keys_num;
    IIIMCF_keyevent *pkev_on;
    int off_keys_num;
    IIIMCF_keyevent *pkev_off;
    char          *engine_name;
    char          *default_font_name;
    char          *primary_locale;
    char          *client_type;
    XIC           current_ic;
    im_locale_pair *lang_alias;
    LangElement   *supported_languages;
    int           count_languages;
    XIMMethods    switch_methods;
    int counter;
} XIMIImpIMRec;

```

_Ximp_OpenIM

Description:

This opens Ximp legacy InputMethod which is for non-UTF-8 Asian locales.

Source file:

`<ximp40.so – XimpIM.c>`

Internal detail:

It allocates struct `Ximp_XIMRec` `<ximp40.so – XimpIm.h>` which can be used as XIM as the first two members are the same.

```

typedef struct _Ximp_XIM {
    XIMMethods    methods;

```

```

XIMCoreRec          core;
XIMXimpRec          *ximp_impact;
} Ximp_XIMRec;

```

_Ximp_OpenIM set methods member as the following.

```

close = _Ximp_CloseIM;           <ximp40.so - XimpIM.c>
set_values = _Ximp_SetIMValues;  <ximp40.so - XimpIM.c>
get_values = _Ximp_GetIMValues;  <ximp40.so - XimpIM.c>
create_ic = _Ximp_CreateIC;      <ximp40.so - XimpIC.c>
ctstombs = _Ximp_ctstombs;       <ximp40.so - XimpConv.c>
ctstowcs = _Ximp_ctstowcs;       <ximp40.so - XimpConv.c>

```

And it initializes `ximp_impact` (`XIMXimpRec` [<ximp40.so - XimpIm.h>](#)).

_XimpLocalOpenIM

Description:

This opens local InputMethod for non UTF-8 EMEA locales. Local means it does not connect to InputMethod server, and all of composition work is done in `xiiimp.so` library itself.

Sourced file:

[<xiiimp.so - composeIM.c>](#)

Internal detail:

It allocates struct `XimCommonRec` and set methods member as follows.

```

close = _Ximp_Local_CloseIM;     <xiiimp.so - composeIM.c>
set_values = _Ximp_Local_SetIMValues; <xiiimp.so - composeIM.c>
get_values = _Ximp_Local_GetIMValues; <xiiimp.so - composeIM.c>
create_ic = _Ximp_Local_CreateIC;   <xiiimp.so - composeIM.c>
ctstombs = _Ximp_ctstombs;         <xiiimp.so - XimpConv.C>
ctstowcs = _Ximp_ctstowcs;         <xiiimp.so - XimpConv.c>

```

And call **CommonOpenIM** and **COMPOSE_OpenIM_SWITCH** (these are the same in **_SwitchOpenIM**).

_XDefaultOpenIM

Description:

This opens default (fallback) InputMethod for C locale.

Source file: [<libX11.so - XSunIMIF.c>](#)

Internal detail:

It allocates struct `StaticXIMRec` [<libX11.so - XSunIMIF.c>](#) as XIM structure.

```

typedef struct _StaticXIM {

```

```

    XIMMethods          methods;
    XIMCoreRec          core;
    XIMStaticXIMRec    *private;
} StaticXIMRec;

```

And set methods member as follows.

```

close = _CloseIM;           <libX11.so – XSunIMIF.c>
set_values = _SetIMValues;  <libX11.so – XSunIMIF.c>
get_values = _GetIMValues;  <libX11.so – XSunIMIF.c>
create_ic = _CreateIC;      <libX11.so – XSunIMIF.c>
ctstombs = NULL;
ctstowcs = NULL;

```

It also initialize private member with XIMStaticXIMRec <libX11.so – XSunIMIF.c>

```

typedef struct _XIMStaticXIMRec {
    /* for CT => MB,WC converter */
    XlcConv          ctom_conv;
    XlcConv          ctow_conv;
} XIMStaticXIMRec;

```

XGetIMValues

Arguments:

```

XIM im;
...

```

Returns:

Name of failed to get attribute, otherwise NULL

Description:

This API can be used for getting IM supported input style (attribute name is `XNQueryInputStyle` and its value is: `XIMStyles *`). The `XIMStyles` determine how IM send data to application regarding preedit string and status information. It depends on application which `XIMStyles` (that are returned with this API) are used. The application specify input method style at **XCreateIC**.

Source file:

<libX11 – ICWrap.c>

Internal details:

It calls `get_values` function in `XIMMethods` of `XIM` structure. So returned input style values depends on **XOpenIM** body functions. In other words, it depends on user's locale.

UTF-8 locale (_SwitchOpenIM) case:

This calls **IIIMP_GetIMValues** <iiimp.so – iiimpIM.c>. This function supports some extended attributes not only `XNQueryInputStyle`. The below attributes are handled in this function.

XNQueryInputStyle:

Returns all of combination of preedit and status styles. This means if the application runs in UTF-8 locales on Solaris, all of preedit/status styles are available for the application. They are

```
static XIMStyle im_styles[] = {
    XIMPreeditCallbacks | XIMStatusCallbacks,
    XIMPreeditCallbacks | XIMStatusArea,
    XIMPreeditCallbacks | XIMStatusNothing,
    XIMPreeditPosition | XIMStatusCallbacks,
    XIMPreeditPosition | XIMStatusArea,
    XIMPreeditPosition | XIMStatusNothing,
    XIMPreeditArea | XIMStatusCallbacks,
    XIMPreeditArea | XIMStatusArea,
    XIMPreeditArea | XIMStatusNothing,
    XIMPreeditNothing | XIMStatusCallbacks,
    XIMPreeditNothing | XIMStatusArea,
    XIMPreeditNothing | XIMStatusNothing,
    XIMPreeditCallbacks | XIMStatusNone,
    XIMPreeditPosition | XIMStatusNone,
    XIMPreeditArea | XIMStatusNone,
    XIMPreeditNothing | XIMStatusNone,
    XIMPreeditNone | XIMStatusCallbacks,
    XIMPreeditNone | XIMStatusArea,
    XIMPreeditNone | XIMStatusNothing,
    XIMPreeditNone | XIMStatusNone,
};
```

XNMultiLingualInput:

This non-public attribute is used by OpenOffice.org implementation for multilingual input. This attribute is not used by other than OpenOffice.org. It returns true if **__XOpenIM** <[xiiimp.so](#) – [XeIMWrap.c](#)> (private API for OpenOffice.org only) specifies it is multilingual, otherwise it returns false.

XNQueryExtensionVersion:

This is also non-public attribute, and returns XIIIMP_MULTILINGUAL_EXTENSION_VERSION (2) in current implementation. No user is found for now.

XNQueryUnicodeCharacterSubset:

This is also non-public attribute, and used by OpenOffice.org only.

non UTF-8 Asian locale (_Ximp_OpenIM) case:

This calls **_Ximp_GetIMValues** <[ximp40.so](#) – [XimpIM.c](#)>. This function supports only public XNQueryInputStyle attribute.

XNQueryInputStyle:

Returns styles set in Ximp_XIMRec structure's ximp_impact. This value depends on XIM server. It's iim-xbe in Solaris case, so it supports all of possible combinations for preedit and status styles.

non UTF-8 EMEA locale (*_XimpLocalOpenIM*) case:

This calls ***_Ximp_Local_GetIMValues*** <*xiiimp.so – composeIM.c*> . This function supports only public `XNQueryInputStyle` attribute.

`XNQueryInputStyle`:

Returns 6 kind of styles listed below.

```
XIMPreeditNone | XIMStatusNone;
XIMPreeditNothing | XIMStatusNothing;
XIMPreeditPosition | XIMStatusArea;
XIMPreeditNone | XIMStatusArea;
XIMPreeditNothing | XIMStatusArea;
XIMPreeditNone | XIMStatusNothing;
```

C locale (*_XDefaultOpenIM*) case:

This calls ***_GetIMValues*** <*libX11.so – XSunIMIF.c*>. This function supports only public `XNQueryInputStyle` attribute.

`XNQueryInputStyle`:

Returns one style

```
XIMPreeditNone | XIMStatusNone
```

That means this IM does not support any preedit or status handling.

XSetIMValues

Arguments:

```
XIM im;
...
```

Returns:

Name of failed to set attribute, otherwise NULL.

Description:

This API can be used to set IM attributes which specified IM supports. But the supported attributes are IM dependent (no IM neutral attribute is defined).

Source file:

<*libX11.so – ICWrap.c*>

Internal details:

This API delegates actual process to XIM methods->set_values function. So it depends on **XOpenIM** body functions. In other words, it depends on user's locale.

UTF-8 locale (*_SwitchOpenIM*) case:

This calls ***IIIMP_SetIMValues*** <*xiiimp.so – iiimpIM.c*>. The supported attributes are the follows.

```
“engineInterfaceName”
“applicationType”
“defaultFontname”
“primaryLocale”
XNDestroyCallback
```

The first three attributes are used by `iiim-xbe` (Solaris version of XIM server which connect to `iiim` server). Other two attributes are not used for now.

non UTF-8 Asian locale (`_Ximp_OpenIM`) case:

This calls `_Ximp_SetIMValues` *<ximp40.so – XimpIM.c>*. This always returns requested attribute name. It means no attribute is supported.

non UTF-8 EMEA locale (`_XimpLocalOpenIM`) case:

This calls `_Ximp_Local_SetIMValues` *<xiiimp.so – composeIM.c>*. This always returns requested attribute name. It means no attribute is supported.

C locale (`_XDefaultOpenIM`) case:

This calls `_SetIMValues` *<libX11.so – XSunIMIF.c>*. This always returns requested attribute name. It means no attribute is supported.

XCloseIM

Arguments:

XIM im;

Returns:

Zero if it fails in some reason, otherwise non-zero.

Description:

Close the im connection which is opened by **XOpenIM**.

Source file:

<libX11.so – IMWrap.c>

Internal detail:

It calls `close` function in `XIMMethods` of XIM structure. So close procedure depends on **XOpenIM** body function. And it also calls `_XCloseLC` which corresponds to `_XOpenLC` at **XOpenIM**. What all of close functions do is freeing allocated resources for its IM.

UTF-8 locale (`_SwitchOpenIM`) case:

This calls `SwitchCloseIM` *<xiiimp.so – switchIM.c>*.

non UTF-8 Asian locale (`_Ximp_OpenIM`) case:

This calls `_Ximp_CloseIM` *<ximp40.so – XimpIM.c>*.

non UTF-8 EMEA locale (`_XimpLocalOpenIM`) case:

This calls `_Ximp_Local_CloseIM` *<xiiimp.so – composeIM.c>*.

C locale (`_XDefaultOpenIM`) case:

This calls `_CloseIM` *<libX11.so – XSunIMIF.c>*


```

XIMStyle          input_style;          /* IM's input style */
Window            focus_window;         /* where key events go */
unsigned long     filter_events;        /* event mask from IM */
XIMCallback       geometry_callback;    /* client callback */
char *            res_name;
char *            res_class;
XIMCallback       destroy_callback;
XIMCallback       string_conversion_callback;
XIMStringConversionText string_conversion;
XIMResetState     reset_state;
XIMHotKeyTriggers *hotkey;
XIMHotKeyState    hotkey_state;
ICPreeditAttributes preedit_attr;      /* visuals of preedit area */
ICStatusAttributes status_attr;        /* visuals of status area */
} XICCoreRec, *XICCore;

typedef struct _XicCommon {
XICMethods        methods;
XICCoreRec        core;
XICXimp           ximp_icpart;
XICPopup          popup_icpart;
XICGUI            gui_icpart;

/* XIC private part */
XICComposeIM     local_icpart; /* must be null when IIIMP only */
XICIIIMP         iiimp_icpart; /* must be null when localIM only */
XIMCallback       switchim_notify_callback;
XIMCallback       commit_string_callback;
XIMCallback       forward_event_callback;
XIMUnicodeCharacterSubsetID subset_id;
XIMCallback       lookup_start_callback;
XIMCallback       lookup_draw_callback;
XIMCallback       lookup_done_callback;
Bool (*active_filter)(Display *, Window, XEvent *, XPointer);
XICMethods        active_methods;
struct _SwitchFilter *switch_filters;
SwitcherContext   *switcher_context; /* per context */
char *current_language;
char *current_le;
int kbd_layout;
Window client_toplevel; /* Save client top level window */
} XicCommonRec;

```

And sets its methods member as follows.

```

destroy = DestroyIC <xiiimp.so - switchIM.c>
set_focus = SetFocus <xiiimp.so - switchIM.c>
unset_focus = UnSetFocus <xiiimp.so - switchIM.c>
set_values = IIIMP_SetICValues <xiiimp.so - iiimpIC.c>
get_values = IIIMP_GetICValues <xiiimp.so - iiimpIC.c>
mb_reset = MbReset <xiiimp.so - switchIM.c>
wc_reset = WcReset <xiiimp.so - switchIM.c>
mb_lookup_string = MbLookupString <xiiimp.so - switchIM.c>
wc_lookup_string = WcLookupString <xiiimp.so - switchIM.c>

```

These functions are the bodies of public API implementations for **XDestroyIC**, **XSetICFocus**, **XUnsetICFocus**, **XSetICValues**, **XGetICValues**, **XmbResetIC**, **XwcResetIC**, **XmbLookupString** and **XwcLookupString** respectively in UTF-8 locales.

Then it calls **CommonCreateIC** <xiiimp.so - commonIM.c>, **COMPOSE_CreateIC_SWITCH** <xiiimp.so - composeIM.c>, **IIIMP_CreateIC_SWITCH** <xiiimp.so - iiimpIC.c> and **ResetSwitchFilter** <xiiimp.so - switchIM.c>.

CommonCreateIC initializes `ximp_icpart` and `gui_icpart`.

COMPOSE_CreateIC_SWITCH initializes `XicCommon local_icpart` and `set local_icpart->switch_methods` as follows.

```

destroy = SWITCH_DestroyIC
set_focus = _Ximp_Local_SetFocus
unset_focus = _Ximp_Local_UnSetFocus

```

And `set local_icpart->imstate` with `XIM local_impact->top_state` which is initialized with `Compose` file in **COMPOSE_OpenIM_SWITCH**. And call **Ximp_Local_Status_Set**, **Ximp_Local_Status_Start** and **Ximp_Local_Status_Draw** <xiiimp.so - status.c> for displaying initial status string. Then call **RegisterSwitchFilter** <xiiimp.so - switchIM.c> to register following `SwitchFilterEventRec` <xiiimp.so - commonIM.h> at the top of `XicCommon switch_filters` list.

```

typedef struct _SwitchFilter {
    struct _SwitchFilter *next;
    SwitchKeyEventProc is_switch_key;
    XFilterEventProc key_filter;
    XICMethods ic_methods;
} SwitchFilterEventRec, *SwitchFilterEventList;

next = NULL;
is_switch_key = SwitchFilter <xiiimp.so - composeIM.c>
key_filter = Ximp_Local_KeyFilter <xiiimp.so - composeIM.c>
ic_methods = {
    destroy = _Ximp_Local_DestroyIC;
        set_focus = _Ximp_Local_SetFocus;
    unset_focus = _Ximp_Local_UnSetFocus;
    set_values = _Ximp_Local_SetICValues;
    get_values = _Ximp_Local_GetICValues;
    mb_reset = _Ximp_Local_MbReset;
    wc_reset = _Ximp_Local_WcReset;
    mb_lookup_string = _Ximp_Local_MbLookupString;
    wc_lookup_string = _Ximp_Local_WcLookupString;
}

```

IIIMP_CreateIC_SWITCH initializes `XicCommon iiimp_icpart` and set `iiimp_icpart->switch_methods` as follows.

```
destroy = SWITCH_DestroyIC
set_focus = IIIMP_SetFocus
unset_focus = IIIMP_UnSetFocus
```

And call **IMCreateIC** [<xiiimp.so – iimcfFun.c>](#) to setup `iiimp_icpart->context` which represents remote iim server input context handle. Then call **RegisterSwitchFilter** [<xiiimp.so – switchIM.c>](#) to add following `SwitchFilterEventRec` to `XicCommon switch_filters`.

```
next = "above data which is registered COMPOSE_CreateIC_SWITCH"
is_switch_filter = SwitchFilter <xiiimp.so – iiimpIC.c>
key_filter = IIIMP_Local_KeyFilter <xiiimp.so – iiimpIC.c>
ic_methods = {
    destroy = IIIMP_DestroyIC;
    set_focus = IIIMP_SetFocus;
    unset_focus = IIIMP_UnSetFocus;
    set_values = IIIMP_SetICValues;
    get_values = IIIMP_GetICValues;
    mb_reset = IIIMP_MbReset;
    wc_reset = IIIMP_WcReset;
    mb_lookup_string = IIIMP_MbLookupString;
    wc_lookup_string = IIIMP_WcLookupString;
}
```

Then call **ResetSwitchFilter** [<xiiimp.so – switchIM.c>](#) to set `XicCommon active_filter` and `active_methods` as follows.

```
active_filter = Ximp_Local_KeyFilter <xiiimp.so – composeIM.c>
active_methods = {
    destroy = _Ximp_Local_DestroyIC;
    set_focus = _Ximp_Local_SetFocus;
    unset_focus = _Ximp_Local_UnSetFocus;
    set_values = _Ximp_Local_SetICValues;
    get_values = _Ximp_Local_GetICValues;
    mb_reset = _Ximp_Local_MbReset;
    wc_reset = _Ximp_Local_WcReset;
    mb_lookup_string = _Ximp_Local_MbLookupString;
    wc_lookup_string = _Ximp_Local_WcLookupString;
}
```

non UTF-8 Asian locale (`_Ximp_OpenIM`) case:

This calls **_Ximp_CreateIC** [<ximp40.so – XimpIC.c>](#). It allocates `Ximp_XICRec` [<ximp40.so – XimpIm.h>](#) which first members are the same with `XIC` structure's members.

```
typedef struct _Ximp_XIC {
    XICMethods    methods;
    XICCoreRec    core;
    XICXimpRec    *ximp_icpart;
} Ximp_XICRec;
```

And initialize `ximp_icpart` and set `XICMethods` as follows.

```

destroy = _Ximp_DestroyIC <ximp40.so – XimpIC.c>
set_focus = _Ximp_SetFocus <ximp40.so – XimpIC.c>
unset_focus = _Ximp_UnSetFocus <ximp40.so – XimpIC.c>
set_values = _Ximp_SetICValues <ximp40.so – XimpICS.c>
get_values = _Ximp_GetICValues <ximp40.so – XimpICG.c>
mb_reset = _Ximp_MbReset <ximp40.so – XimpLkup.c>
wc_reset = _Ximp_WcReset <ximp40.so – XimpLkup.c>
mb_lookup_string = _Ximp_MbLookupString <ximp40.so – XimpLkup.c>
wc_lookup_string = _Ximp_WcLookupString <ximp40.so – XimpLkup.c>

```

non UTF-8 EMEA locale (XimpLocalOpenIM) case:

This calls **Ximp_Local_CreateIC** <xiiimp.so – composeIM.c>. It allocates XicCommonRec and set its XICMethods as follows.

```

destroy = _Ximp_Local_DestroyIC <xiiimp.so – composeIM.c>
set_focus = _Ximp_Local_SetFocus <xiiimp.so – composeIM.c>
unset_focus = _Ximp_Local_UnSetFocus <xiiimp.so – composeIM.c>
set_values = _Ximp_Local_SetICValues <xiiimp.so – composeIM.c>
get_values = _Ximp_Local_GetICValues <xiiimp.so – composeIM.c>
mb_reset = _Ximp_Local_MbReset <xiiimp.so – composeIM.c>
wc_reset = _Ximp_Local_WcReset <xiiimp.so – composeIM.c>
mb_lookup_string = _Ximp_Local_MbLookupString <xiiimp.so – composeIM.c>
wc_lookup_string = _Ximp_Local_WcLookupString <xiiimp.so – composeIM.c>

```

And it registers **Ximp_Local_KeyFilter** <xiiimp.so – composeIM.c> for KeyPress and KeyRelease event by **XRegisterFilterByType** <libX11.so – RegstFlt.c>.

C locale (XDefaultOpenIM) case:

This calls **CreateIC** <libX11.so – XSunIMIF.c>. It allocates XICRec <libX11.so – Xlcint.h> and set its XICMethods member as follows.

```

destroy = _DestroyIC <libX11.so – XSunIMIF.c>
set_focus = _SetFocus <libX11.so – XSunIMIF.c>
unset_focus = _UnsetFocus <libX11.so – XSunIMIF.c>
set_values = _SetICValues <libX11.so – XSunIMIF.c>
get_values = _GetICValues <libX11.so – XSunIMIF.c>
mb_reset = _MbReset <libX11.so – XSunIMIF.c>
wc_reset = _WcReset <libX11.so – XSunIMIF.c>
mb_lookup_string = _MbLookupString <libX11.so – XSunIMIF.c>
wc_lookup_string = _WcLookupString <libX11.so – XSunIMIF.c>

```

XSetICFocus

Arguments:

XIC ic;

Returns:

none

Description:

This API is the trigger to start filtering keyevent to utilize InputMethod for specified InputContext. Application usually calls this API when its text input area get keyboard focus.

Source file:

[<libX11.so – ICWrap.c>](#)

Internal detail:

It calls `set_focus` function in `XICMethods` of `XIC`. So the body of it depends on `XCreateIC` body which is determined by user's locale. But main task for this function is registering filter procedure for keyevent which is called in `XFilterEvent` in any case.

UTF-8 locale (SwitchCreateIC) case:

This calls `SetFocus` [<xiiimp.so – switchIM.c>](#). This function register `Switch_KeyFilter` [<xiiimp.so – switchIM.c>](#) for KeyPress and KeyRelease events by `_XRegisterFilterByType` internal API [<libX11.so – RegstFlt.c>](#). This means `XFilterEvent` calls `Switch_KeyFilter` function when application calls `XFilterEvent` with KeyPress or KeyRelease event.

`SetFocus` function for UTF-8 locales also handle various user properties which are set by `iiim-properties` utility. This is described in internal details for `iiim-properties`.

non UTF-8 Asian locale () case:

This calls `_Ximp_SetFocus` [<ximp40.so – XimpIC.c>](#). It ensures XIM server (`iiim-xbe`) connection (if it's not connected yet, then connect) and registers `_Ximp_XimFilter_Keypress` [<ximp40.so – XimpLkup.c>](#) for KeyPress event and `_Ximp_XimFilter_Keyrelease` [<ximp40.so – XimpLkup.c>](#) for KeyRelease event by `_XRegisterFilterByType` [<libX11.so – RegstFlt.c>](#).

non UTF-8 EMEA locale () case:

This calls `_Ximp_Local_SetFocus` [<xiiimp.so – composeIM.c>](#). It handles showing status string and registers `Ximp_Local_KeyFilter` [<xiiimp.so – composeIM.c>](#) for KeyPress and KeyRelease event by `_XRegisterFilterByType` [<libX11.so – RegstFlt.c>](#).

C locale () case:

This calls `_SetFocus` [<libX11.so – XSunIMIF.c>](#), but it does nothing.

XFilterEvent

Arguments:

```
XEvent *event;  
Window w;
```

Returns:

If event is consumed by InputMethod, then return True, otherwise False.

Description:

This API is the entry point for the application pass the information to InputMethod. Application which want to use InputMethod facility calls this function with XEvent which is just got with `XNextEvent` API.

Source file:

[<libX11.so – FilterEv.c>](#)

Internal details:

This API calls registered procedures by **`_XRegisterFilterByType`** [<libX11.so – RegstFlt.c>](#) if registered procedure's window and event type are equal with this API argument window and event type. The filter procedures are registered by locale dependent **`XSetICFocus`** or **`XCreateIC`** implementation as described in **`XSetICFocus`** or **`XCreateIC`** sections.

In UTF-8 locale, it behave like below.

When key event comes to **`XFilterEvent`**, then **`Switch_KeyFilter`** [<xiiimp.so – switchIM.c>](#) is called. It calls internal `switch_filters` as described in `XCreateIC/UTF-8` case. The `switch_filters` checks incoming event is trigger for InputMethod or not and change `active_filter` member of `XicCommon` [<xiiimp.so – commonIM.h>](#) if it's trigger event. Then call actual filter function set in `active_filter`. In brief description, incoming keyevent is forwarded to `iiimd` server while InputMethod is ON by **`IMForwardEvent`** [<xiiimp.so – iiimcfun.c>](#) in **`IIIMP_Local_KeyFilter`** [<xiiimp.so – iiimpIC.c>](#).

`Xmb/wcLookupString`

Arguments:

```
XIC ic;
XKeyPressedEvent *event;
char *buffer_return;
int bytes_buffer;
KeySym *keysym_return;
Status *status_return;
```

Returns:

Number of bytes in `bytes_buffer` which represents string that application gets

Description:

This API is used for result string which InputMethod processed in its context.

Source file:

[<libX11.so – ICWrap.c>](#)

Internal detail:

It calls argument `XIC`'s `XICMethods->mb/wc_lookup_string` function. This also depends on locale.

UTF-8 locale:

The body is **`MbLookupString/WcLookupString`** [<xiiimp.so – switchIM.c>](#).

non UTF-8 Asian locale:

The body is **`_Ximp_MbLookupString/_Ximp_WcLookupString`** [<ximp40.so – XimpLkup.c>](#)

non UTF-8 EMEA locale:

The body is **`_Ximp_Local_MbLookupString/_Ximp_Local_WcLookupString`** [<xiiimp.so – composeIM.c>](#)

C locale:

The body is **`MbLookupString/_WcLookupString`** [<libX11.so – XSunIMIF.c>](#)

Xmb/wcResetIC

Arguments:

XIC ic;

Returns:

Halfway preedit string or NULL

Description:

This API is used for reset InputContext. If the composing preedit string exists, then it will be disposed.

Source file:

<libX11.so – ICWrap.c>

Internal detail:

It calls argument XIC's `XICMethods->mb/wc_reset` function. This also depends on locale.

UTF-8 locale:

The body is **`MbReset/WcReset`** *<xiiimp.so – switchIM.c>*.

non UTF-8 Asian locale:

The body is **`_Ximp_MbReset/_Ximp_WcReset`** *<ximp40.so – XimpLkup.c>*

non UTF-8 EMEA locale:

The body is **`_Ximp_Local_MbReset/_Ximp_Local_WcReset`** *<xiiimp.so – composeIM.c>*

C locale:

The body is **`MbReset/_WcReset`** *<libX11.so – XSunIMIF.c>*

XGetICValues

Arguments:

XIC ic;
...

Returns:

Name of failed to set attribute, otherwise NULL.

Description:

This API can be used to get IC attributes which are defined in X11 XIM specification. Those include input style, preedit or status attributes, callback functions, etc...

Source file:

<libX11.so – ICWrap.c>

Internal detail:

This calls `get_values` function in `XICMethods` of XIC. So the body of it depends on **`XCreateIC`** function which is determined by user's locale.

UTF-8 locale (SwitchCreateIC) case:

`IIIMP_GetICValues` *<xiiimp.so – iiimpIC.c>* is called. This calls `GetICValueData` *<xiiimp.so –*

iiimpICB.c>.

non UTF-8 Asian locale () case:

_Ximp_GetICValues <ximp40.so – XimpICG.c> is called.

non UTF-8 EMEA locale () case:

_Ximp_Local_GetICValues <xiiimp.so – composeIM.c> is called.

C locale () case:

_GetICValues <libX11.so – XSunIMIF.c> is called. This calls _GetICValueData <libX11.so – XSunIMIF.c>.

XSetICValues

TBD

XUnsetICFocus

TBD

XDestroyIC

TBD

XIMOfIC

TBD

GTK-IM implementation

This section explains the InputMethod implementation for the application which use GTK-IM APIs. Here is the list of InputMethod related APIs in GTK2.0.

gtk_im_multicontesxt_new

gtk_im_

```
GType  gtk_im_context_get_type      (void) G_GNUC_CONST;
void    gtk_im_context_set_client_window (GtkIMContext *context,
GdkWindow *window);
void    gtk_im_context_get_preedit_string (GtkIMContext *context, gchar **str,
                                           PangoAttrList **attrs,
                                           gint *cursor_pos);
gboolean gtk_im_context_filter_keypress (GtkIMContext *context,
```

```

                                GdkEventKey *event);
void gtk_im_context_focus_in      (GtkIMContext *context);
void gtk_im_context_focus_out     (GtkIMContext *context);
void gtk_im_context_reset         (GtkIMContext *context);
void gtk_im_context_set_cursor_location (GtkIMContext *context,
                                GdkRectangle *area);
void gtk_im_context_set_use_preedit (GtkIMContext *context,
                                gboolean use_preedit);
void gtk_im_context_set_surrounding (GtkIMContext *context,
                                const gchar *text,
                                gint len,
                                gint cursor_index);
gboolean gtk_im_context_get_surrounding (GtkIMContext *context,
                                gchar **text,
                                gint *cursor_index);
gboolean gtk_im_context_delete_surrounding (GtkIMContext *context,
                                gint offset,
                                gint n_chars);

```

Low level libraries implementation

Server module implementation

IIIM Utility implementation