How Microsoft Uses C++ to Deliver Office

Huge Size, Small Components

ZACHARY HENKEL

Cppcon
The C++ Conference

2022
September 12th-16th

Welcome to CppCon 2022!

Join #visual_studio channel on CppCon Discord
https://aka.ms/cppcon/discord

- Meet the Microsoft C++ team
- Ask any questions
- Discuss the latest announcements

Take our survey
https://aka.ms/cppcon

Agenda

- Background
- Huge Size
- Small Components
- What’s Next

Background

Background

- 1983: Word for DOS released. Written in C

Background

- 1983: Word for DOS released. Written in C
- 1985: Word for Classic Mac OS released

Background

- 1983: Word for DOS released. Written in C
- 1985: Word for Classic Mac OS released
- 1990: First release of Office suite

Background

- 1983: Word for DOS released. Written in C
- 1985: Word for Classic Mac OS released
- 1990: First release of Office suite
- 1992: PowerPoint moves to C++

Background

- 1983: Word for DOS released. Written in C
- 1985: Word for Classic Mac OS released
- 1990: First release of Office suite
- 1992: PowerPoint moves to C++
- 1995: Common functionality moved to shared library (mso.dll)

Background

- 1983: Word for DOS released. Written in C
- 1985: Word for Classic Mac OS released
- 1990: First release of Office suite
- 1992: PowerPoint moves to C++
- 1995: Common functionality moved to shared library (mso.dll)
- 1996: Office moves to C++

Background

- 1983: Word for DOS released. Written in C
- 1985: Word for Classic Mac OS released
- 1990: First release of Office suite
- 1992: PowerPoint moves to C++
- 1995: Common functionality moved to shared library (mso.dll)
- 1996: Office moves to C++
- 2001: First release for macOS (OS X)

Background

- 1983: Word for DOS released. Written in C
- 1985: Word for Classic Mac OS released
- 1990: First release of Office suite
- 1992: PowerPoint moves to C++
- 1995: Common functionality moved to shared library (mso.dll)
- 1996: Office moves to C++
- 2001: First release for macOS (OS X)
- 2010: Office for the Web

Background

- 1983: Word for DOS released. Written in C
- 1985: Word for Classic Mac OS released
- 1990: First release of Office suite
- 1992: PowerPoint moves to C++
- 1995: Common functionality moved to shared library (mso.dll)
- 1996: Office moves to C++
- 2001: First release for macOS (OS X)
- 2010: Office for the Web
- 2013: Office on iOS & Android

Background

- 1983: Word for DOS released. Written in C
- 1985: Word for Classic Mac OS released
- 1990: First release of Office suite
- 1992: PowerPoint moves to C++
- 1995: Common functionality moved to shared library (mso.dll)
- 1996: Office moves to C++
- 2001: First release for macOS (OS X)
- 2010: Office for the Web
- 2013: Office on iOS & Android
- 2019: Clang analysis for Windows code

Windows

- Win32 Client, Store, Server
- arm, arm64, chpe, x64, x86
- MSVC & clang (only front end)

- iOS vs macOS
- Intel vs Apple silicon
- clang (via XCode)

android

- arm, arm64, x64, x86
- clang (via NDK)

Huge Size

Office Monorepo

- Nearly 350 million lines of code
- Roughly 100 million lines of native code
- 2 check-ins/minute at peak times
- Approximately 4,000 active engineers
- Full set of Office releases is around 50TB

How Many Lines of Code?

How Many Lines of Code?

```
void DisplayPicture(std::string_view file)
{
#if defined(SERVER)
    RenderIMG(file);
#elif defined(CLIENT)
    HDC hdc = GetDC(MainWindow());
    Gdiplus::Graphics graphics(hdc);
    Image image(file);
    graphics.DrawImage(&image);
#endif
}
```

How Many Lines of Code?

```
#ifdef DEBUG
    // Count comparisons for perf
    long m_cComparisons;
#endif

#ifdef DEBUG
#include "printdebugsettings.h"
#endif
```

Alternative Measure for C++

Alternative Measure for C++

- Ideal measure: unique translation units

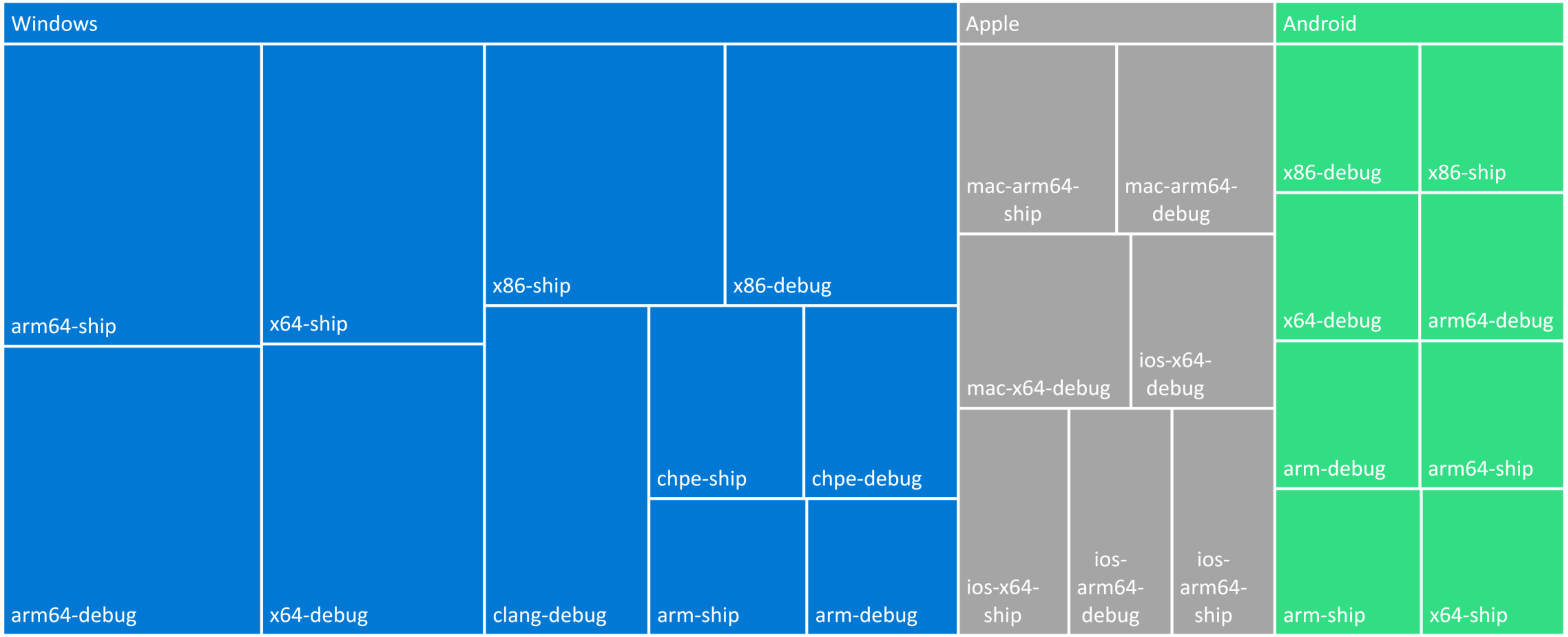
Alternative Measure for C++

- Ideal measure: unique translation units
- Proxy: count compilations

Alternative Measure for C++

- Ideal measure: unique translation units
- Proxy: count compilations
- Office split based on 3 axis:
    - Platform
    - Architecture
    - Debug/Ship

Total Office compiler invocations: 2 Million



Costs of size

Costs of size

- Workload

Costs of size

- Workload
- Static Analysis

Costs of size

- Workload
- Static Analysis
- Migration scope

Costs of size

- Workload
- Static Analysis
- Migration scope
- Tests

Costs of size

- Workload
- Static Analysis
- Migration scope
- Tests
- Decommissioning

Value of expanding size

- 2021: 64-Bit Office for Windows on ARM
- 2020: Office support for Apple silicon
- 2019: Clang analysis for Windows code
- 2015: Office for Windows Store
- 2013: Office on iOS & Android
- 2010: Office for the Web
- 2001: First release for macOS (OS X)

Is it valuable to have a huge codebase?

Is it valuable to have a huge codebase?

It depends

Is it valuable to have a huge codebase?

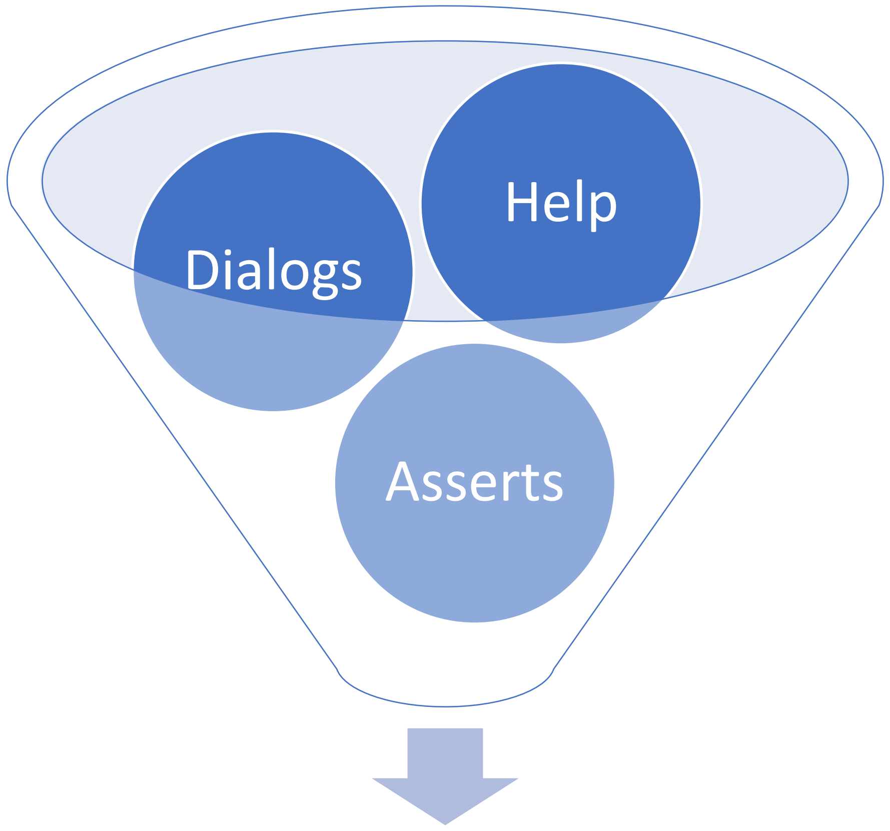
It depends

Is it valuable to have a huge codebase?

It depends

Small Components

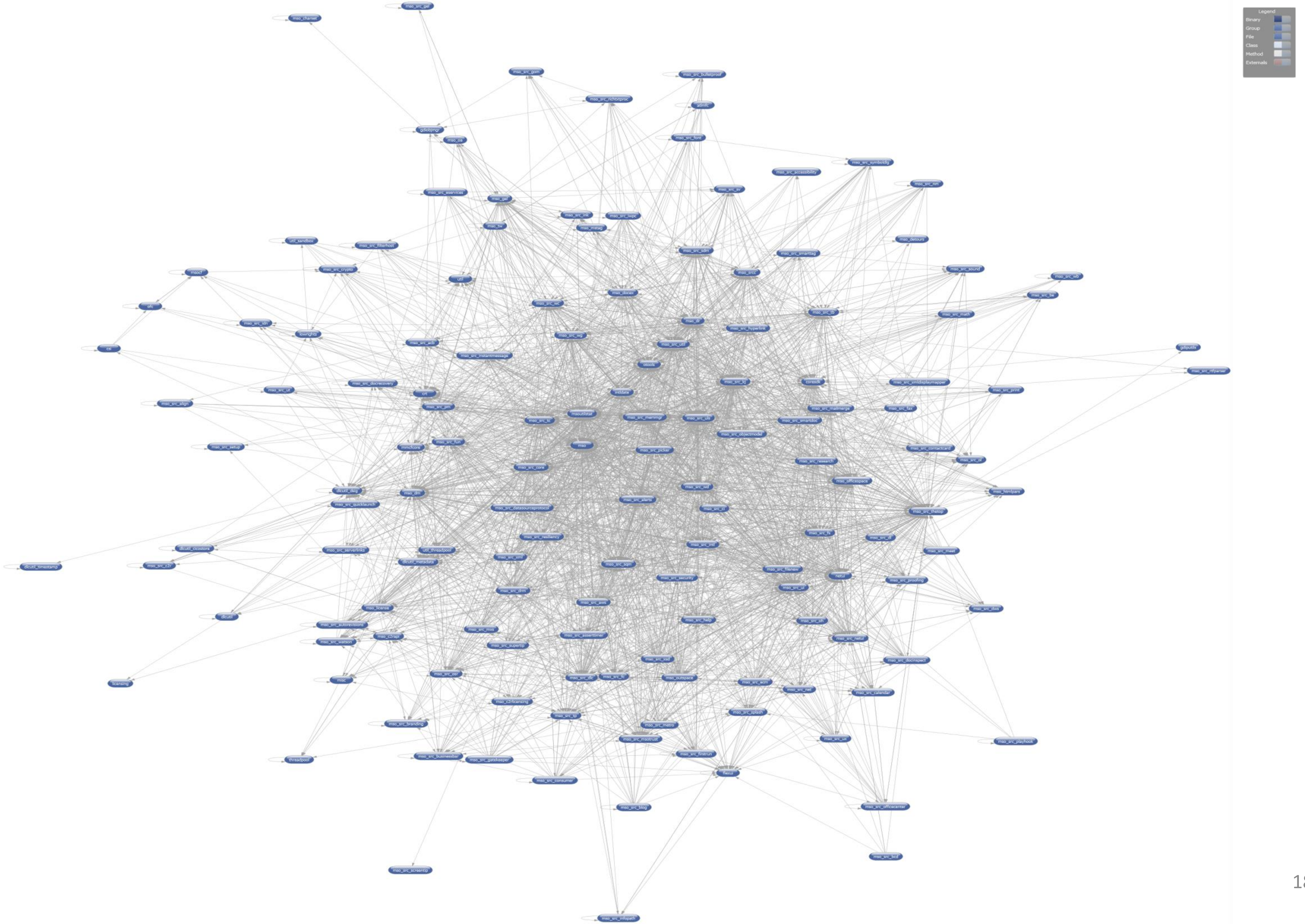
1995: Common functionality moved to shared library



MSO.dll

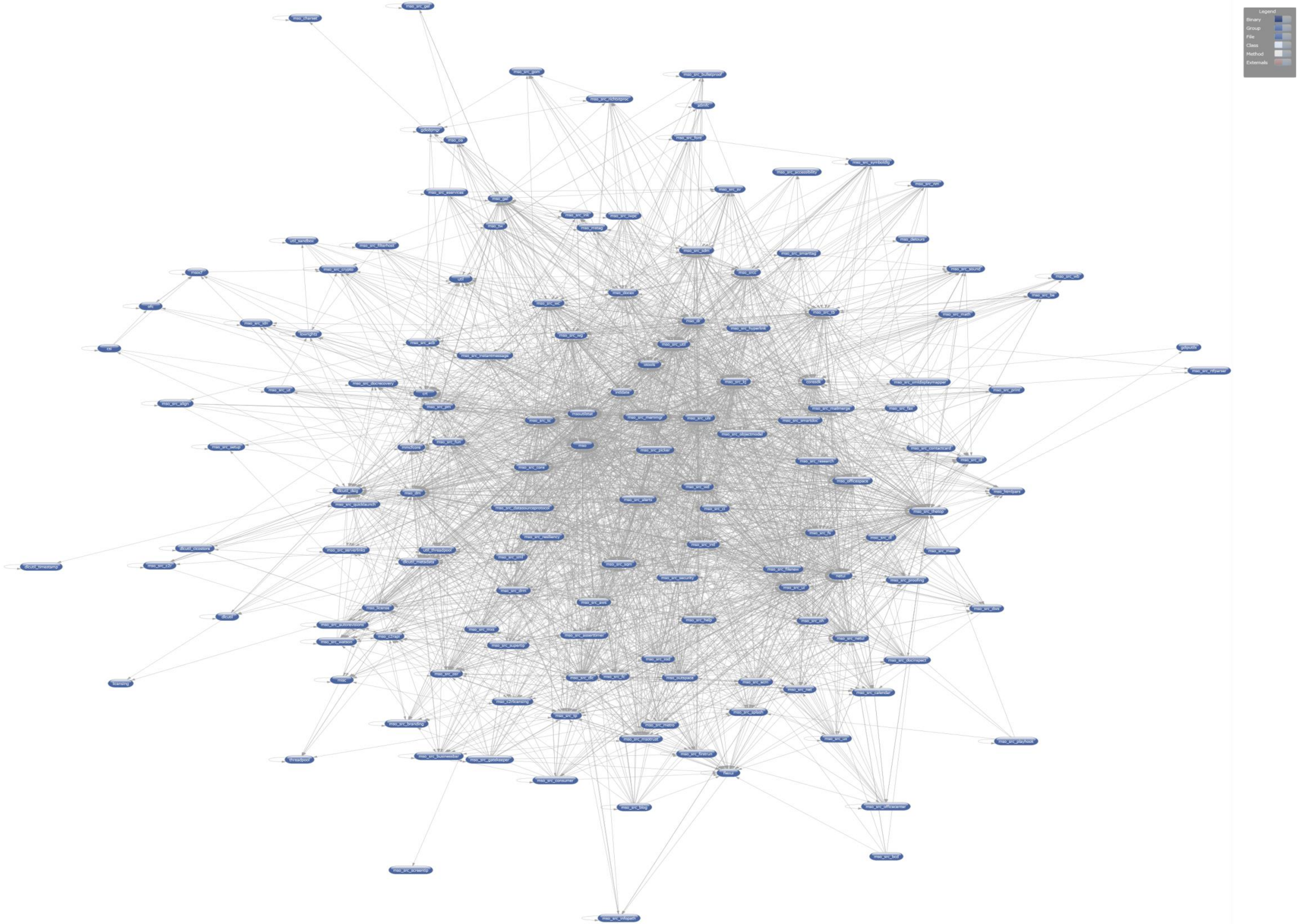
1995: Common functionality moved to shared library

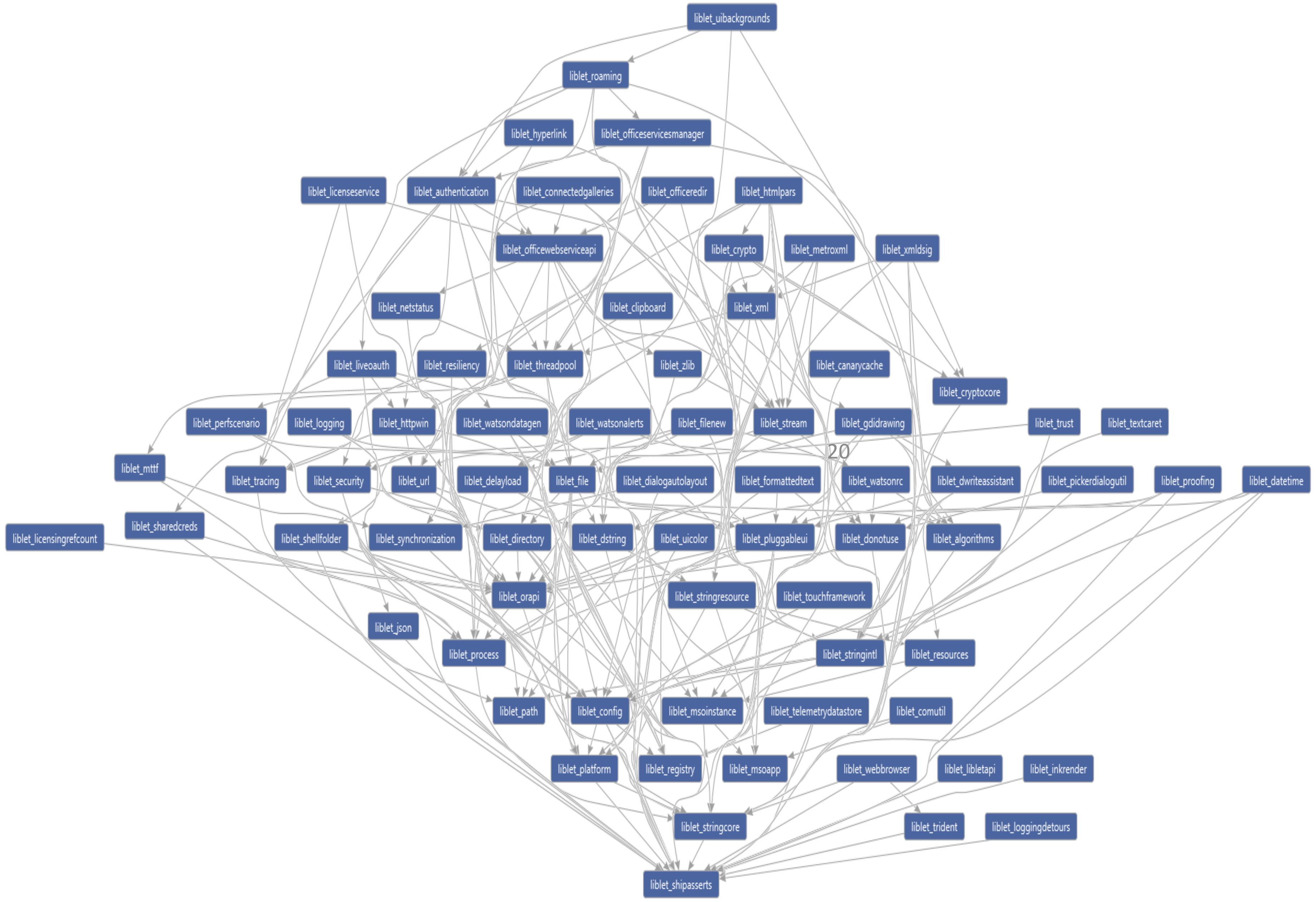
Office 2010



Office 2010

Liblets to the rescue!


Legend
Binary
Group
File
Class
Method
Externals
mso_src_gel
mso_charset
mso_src_gsm
mso_src_bulletproof
mso_src_richtxtproc
atlmfc
mso_src_font
gdiobjmgr
mso_os
mso_src_symboldlg
mso_src_accessibility
mso_src_ism
mso_src_eservices
mso_gel
mso_src_av
mso_src_ink
mso_src_iwpc
mso_tw
mso_metag
mso_detours
util_sandbox
mso_src_filterhost
mso_src_adm
mso_src_smarttag
mso_src_sound
msocf
mso_src_crypto
util
mso_src
mso_src_wb
mso_docex
mso_src_be
mso_src_xc
olc
mso_src_math
mso_src_idn
mso_src_td
mso_ol
lowrights
mso_src_reg
mso_src_hyperlink
mso_src_util
mso_src_adb
csi
gdiputils
mso_src_instantmessage
ostools
mso_src_rtfparser
mso_src_docrecovery
mso_src_lc
coreslk
mso_src_smldisplaymapper
mso_src_ul
mso_src_print
crt
intldata
mso_src_align
mso_src_pm
mso_src_mailmerge
mso_src_fax
msoutlotal
mso_src_mainmgr
mso_src_tc
mso_src_uis
mso_src_smartdoc
mso_src_objectmodel
mso_src_fun
mso
mso_src_setup
mmcfcore
mso_src_picker
mso_src_contactcard
mso_src_research
mso_src_ol
mso_src_core
mso_src_officespace
dicutil_dsig
mso_dm
mso_src_iwd
mso_src_htmlpars
mso_src_alerts
mso_src_quicklaunch
mso_src_datasourceprotocol
mso_src_cl
mso_src_thetop
mso_src_fs
mso_src_resiliency
mso_src_dl
mso_src_inst
dicutil_ocostore
mso_src_serverlinks
util_threadpool
mso_src_meet
dicutil_timestamp
mso_src_xml
dicutil_mscxcore
mso_src_c2r
mso_src_sqm
mso_src_filenew
netui
mso_src_security
mso_src_proofing
mso_src_ui
mso_src_drm
mso_license
mso_src_aws
mso_src_dws
dicutil
mso_src_help
mso_src_ph
mso_src_autoprovisiont
mso_src_mso
mso_c2rapi
mso_src_assertimer
mso_src_netui
mso_src_watson
mso_src_supertip
mso_src_docinspect
mso_src_xsd
mso_src_osi
licensing
msc
mso_src_dlc
mso_src_fc
mso_outspace
mso_src_ecm
mso_src_net
mso_src_calendar
mso_c2rlicensing
mso_src_tp
mso_src_splash
mso_src_branding
mso_src_metro
mso_src_msotrust
mso_src_uss
mso_src_playhook
mso_src_firstrun
threadpool
mso_src_businessbar
mso_src_gatekeeper
flexui
mso_src_consumer
mso_src_blog
mso_src_officecenter
mso_src_bcd
mso_src_screentip
mso_src_infopath


liblet_uibackgrounds
liblet_roaming
liblet_hyperlink
liblet_officeservicesmanager
liblet_licenseservice
liblet_authentication
liblet_connectedgalleries
liblet_officeredir
liblet_htmlpars
liblet_officewebserviceapi
liblet_crypto
liblet_metroxml
liblet_xmldsig
liblet_netstatus
liblet_clipboard
liblet_xml
liblet_liveoauth
liblet_resiliency
liblet_threadpool
liblet_zlib
liblet_canarycache
liblet_cryptocore
liblet_perfscenario
liblet_logging
liblet_httpwin
liblet_watsondatagen
liblet_watsonalerts
liblet_filenew
liblet_stream
liblet_gdidrawing
liblet_trust
liblet_textcaret
liblet_mttf
liblet_tracing
liblet_security
liblet_url
liblet_delayload
liblet_file
liblet_dialogautolayout
liblet_formattedtext
liblet_watsonrc
liblet_dwriteassistant
liblet_pickerdialogutil
liblet_proofing
liblet_datetime
liblet_sharedcreds
liblet_licensingrefcount
liblet_shellfolder
liblet_synchronization
liblet_directory
liblet_dstring
liblet_uicolor
liblet_pluggableui
liblet_donotuse
liblet_algorithms
liblet_orapi
liblet_stringresource
liblet_touchframework
liblet_json
liblet_process
liblet_stringintl
liblet_resources
liblet_path
liblet_config
liblet_msoinstance
liblet_telemetrydatastore
liblet_comutil
liblet_platform
liblet_registry
liblet_msoapp
liblet_webbrowser
liblet_libletapi
liblet_inkrender
liblet_stringcore
liblet_trident
liblet_loggingdetours
liblet_shipasserts

DEN HARTOGH
InterBulk
P&O FERRYMASTERS
UNIT45
20
samskip
P&O FERRYMASTERS
P&O Ferrymasters
P&O Ferrymasters
InterBulk
MARUS

Layers

Layers

mso	• Non-libletized code
mso98	• Clean liblets
mso40ui	• UI Frameworks, Graphics
mso30	• Document synchronization: Authentication, File I/O, Identity
mso20	• Core functionality: Diagnostics, Experimentation, Telemetry

Layers

mso	• Non-libletized code
mso98	• Clean liblets
mso40ui	• UI Frameworks, Graphics
mso30	• Document synchronization: Authentication, File I/O, Identity
mso20	• Core functionality: Diagnostics, Experimentation, Telemetry

Layers

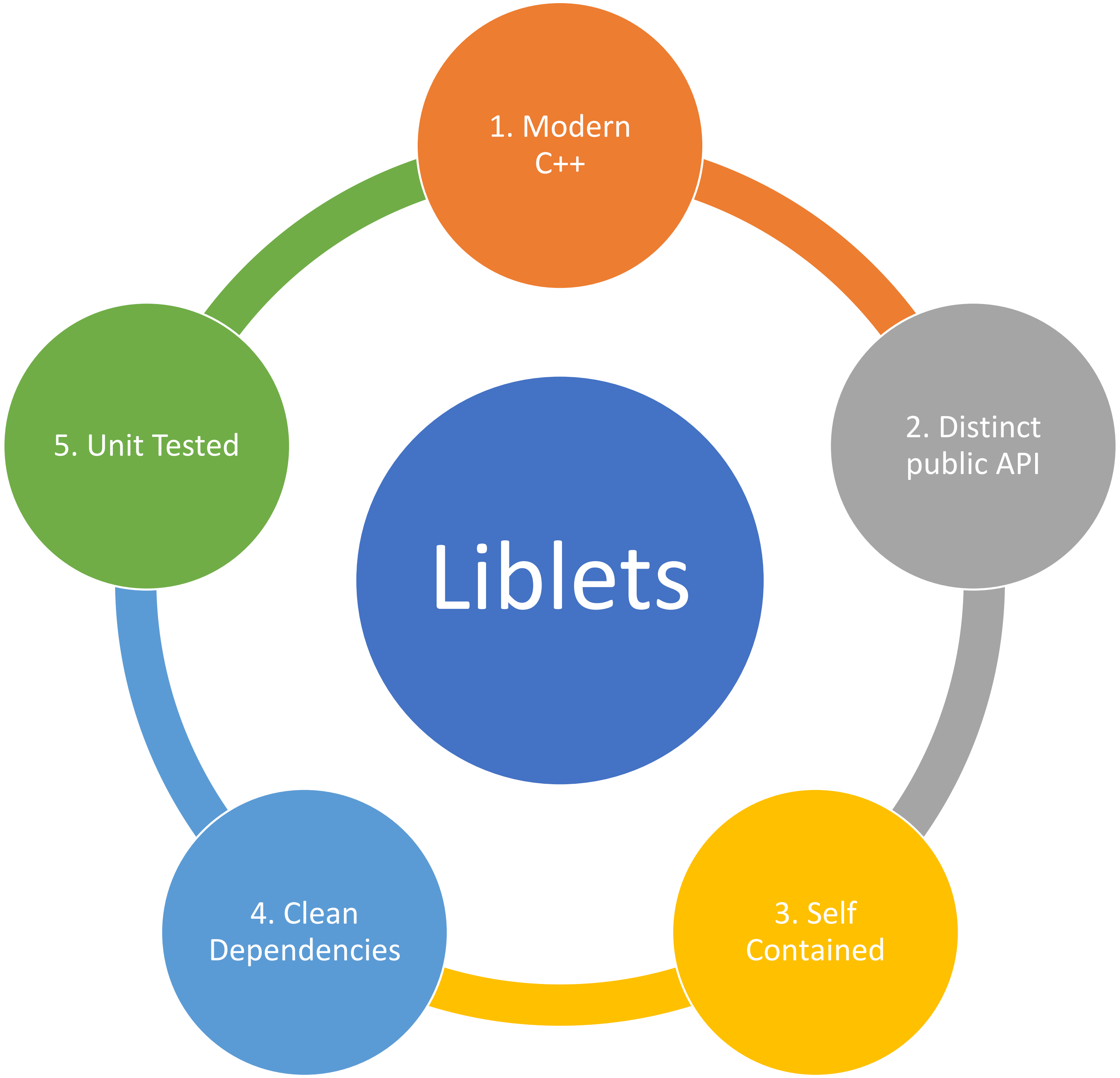
mso	• Non-libletized code
mso98	• Clean liblets
mso40ui	• UI Frameworks, Graphics
mso30	• Document synchronization: Authentication, File I/O, Identity
mso20	• Core functionality: Diagnostics, Experimentation, Telemetry

Layers

mso	• Non-libletized code
mso98	• Clean liblets
mso40ui	• UI Frameworks, Graphics
mso30	• Document synchronization: Authentication, File I/O, Identity
mso20	• Core functionality: Diagnostics, Experimentation, Telemetry

Layers

mso	• Non-libletized code
mso98	• Clean liblets
mso40ui	• UI Frameworks, Graphics
mso30	• Document synchronization: Authentication, File I/O, Identity
mso20	• Core functionality: Diagnostics, Experimentation, Telemetry


Liblets
1. Modern C++
2. Distinct public API
3. Self Contained
4. Clean Dependencies
5. Unit Tested

Liblets use **Modern C++**

- Liblet effort kicked off in 2011
- Exception safe code
- Opened the door for STL usage

Liblets have **A Distinct Public API**

- Headers must explicitly be marked for public consumption
  - Enforced by the build system
- Each public header must be self-contained
- Public APIs in a header must be marked as such

```
class Process
{
LIBLET_PUBLICAPI std::string GetAppPath();

LIBLET_PUBLICAPI_APPLE std::string GetPayloadFolder();

LIBLET_PUBLICAPI_EX("win") std::string GetResFolder(std::string_view lang);
}
```

Symbol visibility

Symbol visibility

```
#ifdef _EXPORTING
   #define CLASS_DECLSPEC    __declspec(dllexport)
#else
   #define CLASS_DECLSPEC    __declspec(dllimport)
#endif
```

Symbol visibility

```
#ifdef _EXPORTING
   #define CLASS_DECLSPEC    __declspec(dllexport)
#else
   #define CLASS_DECLSPEC    __declspec(dllimport)
#endif
```

```
__attribute__((visibility("default")))
__attribute__((visibility("hidden")))
```

Symbol visibility

```
#ifdef _EXPORTING
   #define CLASS_DECLSPEC    __declspec(dllexport)
#else
   #define CLASS_DECLSPEC    __declspec(dllimport)
#endif
```

```
__attribute__((visibility("default")))
__attribute__((visibility("hidden")))
```

```
?Count@?$RoamingList@PEB_W@Roaming@@UEBAKPEBUIOfficeIdentity@Authentication@Mso@@@Z
?Count@?$RoamingList@U_GUID@@@Roaming@@UEBAKPEBUIOfficeIdentity@Authentication@Mso@@@Z
?DeleteItem@?$RoamingList@PEB_W@Roaming@@UEAAJPEBUIOfficeIdentity@Authentication@Mso@@QEB_W@Z
?DeleteItem@?$RoamingList@U_GUID@@@Roaming@@UEAAJPEBUIOfficeIdentity@Authentication@Mso@@U_GUID@@@Z
?InsertItem@?$RoamingList@PEB_W@Roaming@@UEAAJPEBUIOfficeIdentity@Authentication@Mso@@QEB_W_KPEB_WK@Z
?InsertItem@?$RoamingList@U_GUID@@@Roaming@@UEAAJPEBUIOfficeIdentity@Authentication@Mso@@U_GUID@@_KPEB_WK@Z
?MaxCount@?$RoamingList@PEB_W@Roaming@@UEBAKXZ
?MaxCount@?$RoamingList@U_GUID@@@Roaming@@UEBAKXZ
?ReadList@?$RoamingList@PEB_W@Roaming@@UEAAJPEBUIOfficeIdentity@Authentication@Mso@@AEAPEAU?$ListItem@PEB_WPEB_W@2@AEAK@Z
?ReadList@?$RoamingList@U_GUID@@@Roaming@@UEAAJPEBUIOfficeIdentity@Authentication@Mso@@AEAPEAU?$ListItem@U_GUID@@PEB_W@2@AEAK@Z
?Reset@?$RoamingList@PEB_W@Roaming@@UEAAJPEBUIOfficeIdentity@Authentication@Mso@@@Z
?Reset@?$RoamingList@U_GUID@@@Roaming@@UEAAJPEBUIOfficeIdentity@Authentication@Mso@@@Z
```

LIBLET_PUBLICAPI

```
#if defined(APPLE)
    #define LIBLET_PUBLICAPI __attribute__((visibility("default")))
    #define LIBLET_PUBLICAPI_EX(...)
#elif defined(__clang__)
    #define LIBLET_PUBLICAPI __attribute__((annotate("LibletPublicAPI()")))
    #define LIBLET_PUBLICAPI_EX(...) __attribute__((annotate("LibletPublicAPI("#__VA_ARGS__")")))
#else
    #define LIBLET_PUBLICAPI
    #define LIBLET_PUBLICAPI_EX(...)
#endif
```

Liblets are **Self Contained**

- A liblet may have *multiple implementations*
- Each implementation is organized around functionality
  - empty, mock, stub
  - mobile, server
- Architectures are orthogonal

Example implementations

```
void DisplayPicture(std::string_view file)
{
#if defined(SERVER)
    RenderHTML(file);
#elif defined(CLIENT)
    HDC hdc = GetDC(MainWindow());
    Gdiplus::Graphics graphics(hdc);
    Image image(file);
    graphics.DrawImage(&image);
#endif
}
```

Example implementations

gdiimpl.cpp

```
void DisplayPicture(std::string_view file)
{
    HDC hdc = GetDC(MainWindow());
    Gdiplus::Graphics graphics(hdc);
    Image image(file);
    graphics.DrawImage(&image);
}
```

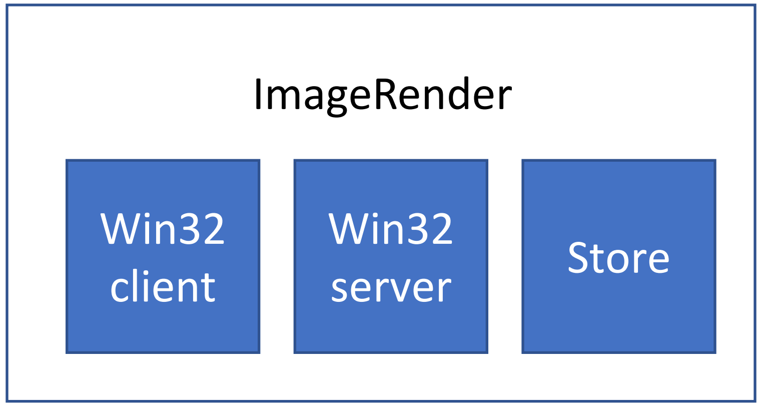
htmlimpl.cpp

```
void DisplayPicture(std::string_view file)
{
    RenderIMG(filename);
}
```

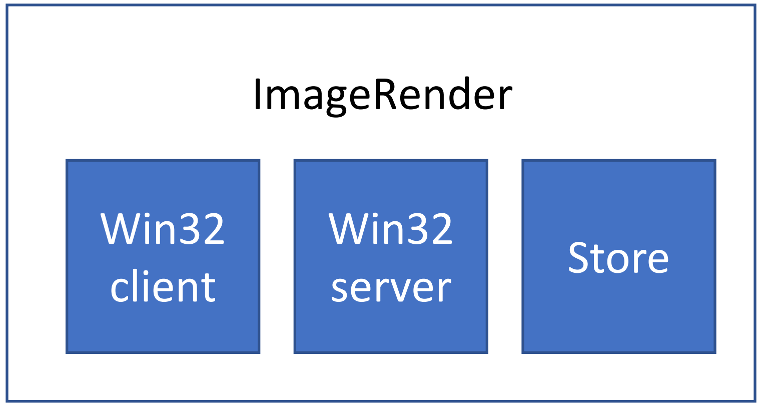
emptyimpl.cpp

```
void DisplayPicture(std::string_view)
{
}
```

Liblets have **Clean Dependencies**

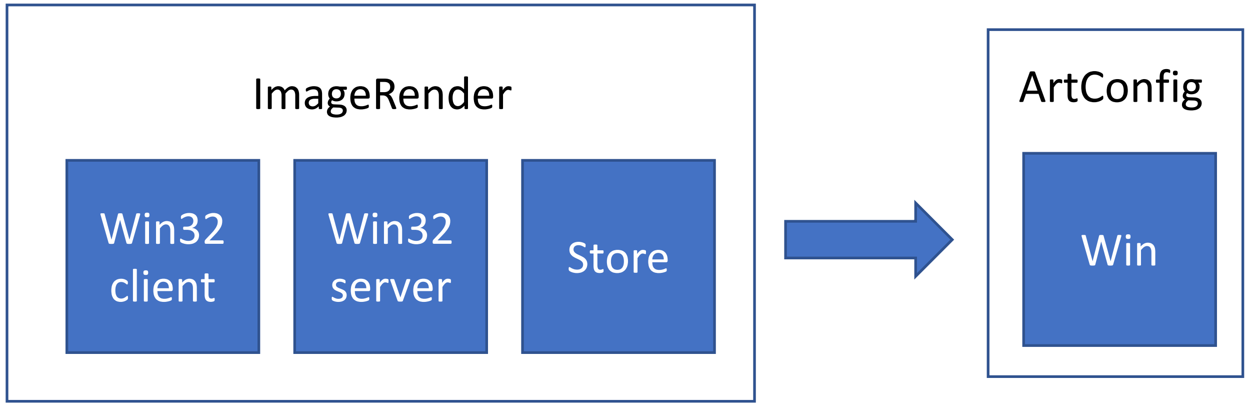


Liblets have **Clean Dependencies**



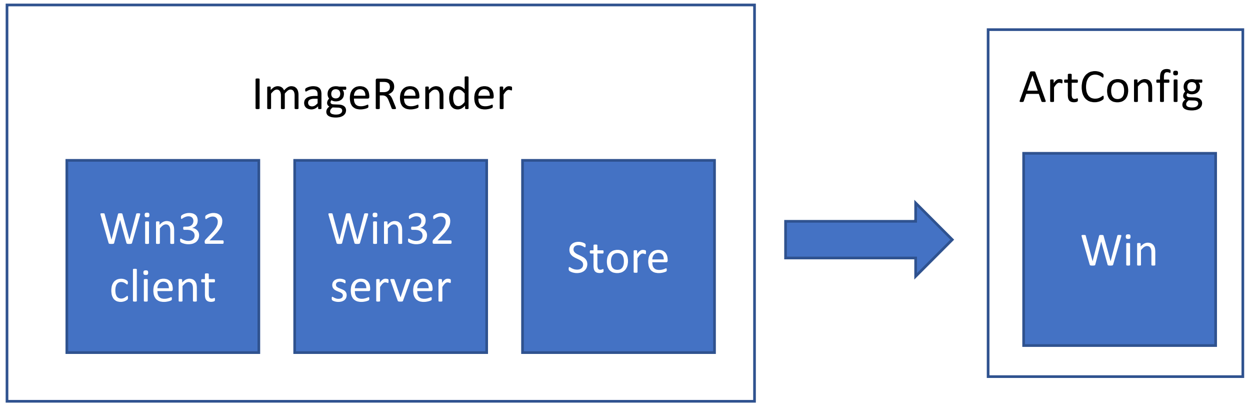
```
<liblet name="ImageRender">
  <dependsOn name="ArtConfig">
  <endpoint name="win32client"/>
  <endpoint name="win32server"/>
  <endpoint name="store"/>
</liblet>
```

Liblets have **Clean Dependencies**



```
<liblet name="ImageRender">
  <dependsOn name="ArtConfig">
  <endpoint name="win32client"/>
  <endpoint name="win32server"/>
  <endpoint name="store"/>
</liblet>
```

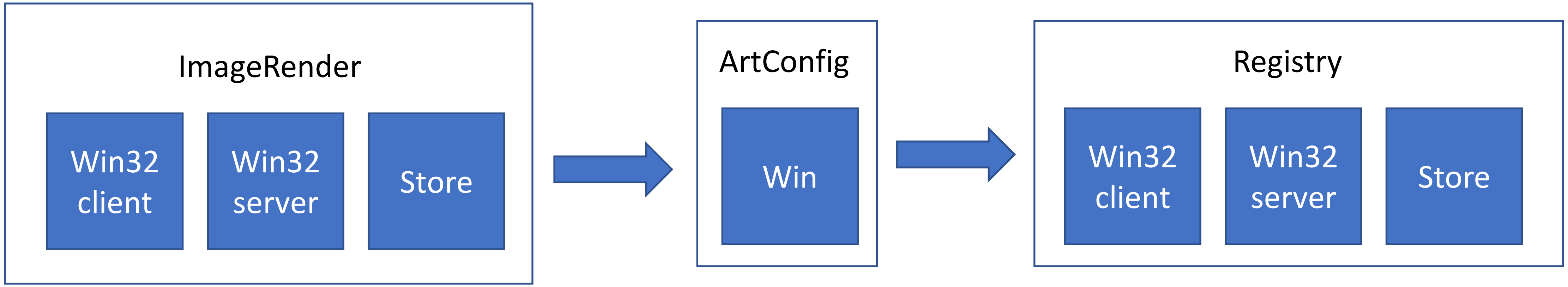
Liblets have **Clean Dependencies**



```
<liblet name="ImageRender">
  <dependsOn name="ArtConfig">
  <endpoint name="win32client"/>
  <endpoint name="win32server"/>
  <endpoint name="store"/>
</liblet>
```

```
<liblet name="ArtConfig">
  <dependsOn name="Registry">
  <endpoint name="win" />
</liblet>
```

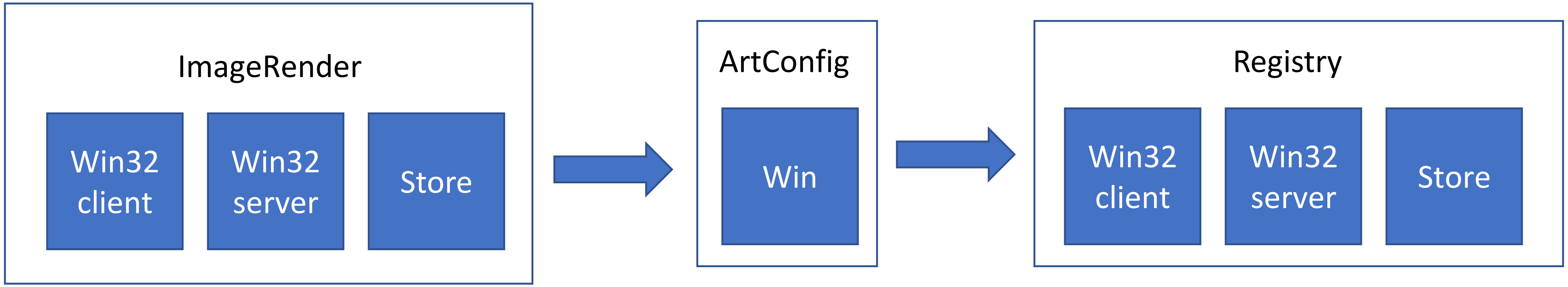
Liblets have **Clean Dependencies**



```
<liblet name="ImageRender">
  <dependsOn name="ArtConfig">
  <endpoint name="win32client"/>
  <endpoint name="win32server"/>
  <endpoint name="store"/>
</liblet>
```

```
<liblet name="ArtConfig">
  <dependsOn name="Registry">
  <endpoint name="win" />
</liblet>
```

Liblets have **Clean Dependencies**

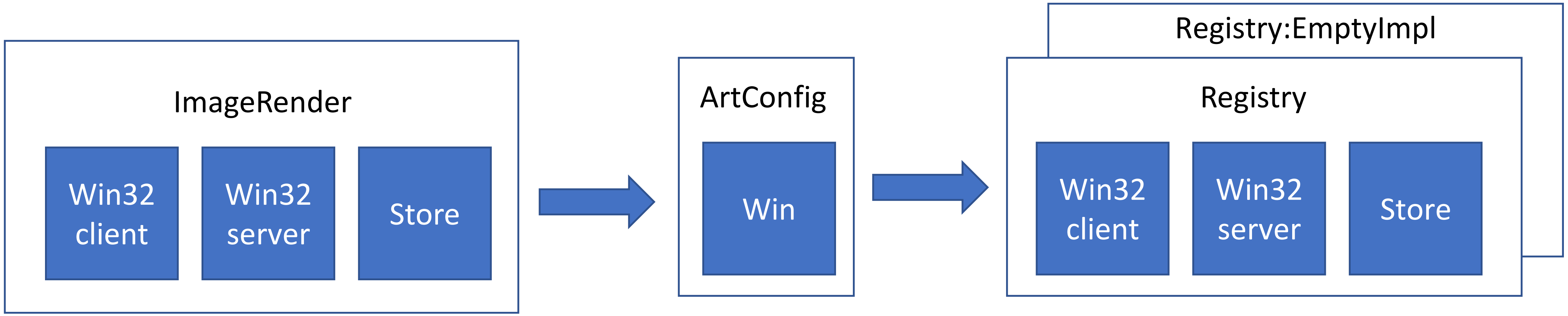


```
<liblet name="ImageRender">
  <dependsOn name="ArtConfig">
  <endpoint name="win32client"/>
  <endpoint name="win32server"/>
  <endpoint name="store"/>
</liblet>
```

```
<liblet name="ArtConfig">
  <dependsOn name="Registry">
  <endpoint name="win" />
</liblet>
```

```
<liblet name="Registry">
<endpoint name="win32client" />
  <endpoint name="win32server" />
  <endpoint name="store" />
</liblet>
```

Liblets have **Clean Dependencies**



```
<liblet name="ImageRender">
  <dependsOn name="ArtConfig">
  <endpoint name="win32client"/>
  <endpoint name="win32server"/>
  <endpoint name="store"/>
</liblet>
```

```
<liblet name="ArtConfig">
  <dependsOn name="Registry">
  <endpoint name="win" />
</liblet>
```

```
<liblet name="Registry">
 <impl>
  <endpoint name="win32client" />
  <endpoint name="win32server" />
  <endpoint name="store" />
 </impl>
 <impl name="EmptyImpl" />
</liblet>
```

Liblets have **Clean Dependencies**

```
# mso40ui Win32 client dll dependencies
LINK_TARGETS = ArtConfig, ImageRender, …
DLL_DEPENDENCIES = mso30win32client.dll, …

# mso40ui Win32 server dll dependencies
LINK_TARGETS = ArtConfig, ImageRender, …
DLL_DEPENDENCIES = mso30win32server.dll, …

# ImageRender test dll
LINK_TARGETS = ArtConfig, ImageRender, Registry:EmptyImpl, …
```

Dependency validation

Dependency validation

- Validation using def files, not the linker

Dependency validation

- Validation using def files, not the linker
- Proof that only public APIs are in use

Dependency validation

- Validation using def files, not the linker
- Proof that only public APIs are in use
- Prevent cycles

Liblets are **Unit Tested**

- Automatically generated mocks

```
TEST_METHOD(TestMocks)
{
    auto mockDoc = Mso::Make<Csi::MockIDocument>();
    mockDoc->mock_IsOpen.returns(false);

    auto mockDocDesc = Mso::Make<MOX::MockIApplicationDocumentDescriptor>();
    mockDocDesc->mock_GetIDocument.returns(mockDoc);

    TestAssert::IsFalse(mockDoc->IsOpen(), L"Testing document->IsOpen(). Expecting false");
    TestAssert::AreEqual(mockDoc.Get(), mockDocDesc->GetIDocument().Get());
}
```

Liblets are **Unit Tested**

```
TEST_METHOD(TestMocks2)
{
    PersisterBase base;
    auto mockDoc = Mso::Make<Csi::MockIDocument>();
    base->doc = &mockDoc;

    int calls = 0;
    bool setDirtyArg = false;

    // SetDirty should also call SetDirty on the child document
    mockDoc.mock_SetDirty = [&calls, &setDirtyArg](bool dirty) noexcept
    {
        ++calls;
        setDirtyArg = dirty;
    };

    base->SetDirty(true);
    TestAssert::AreEqual(calls, 1);
    TestAssert::IsTrue(setDirtyArg);
}
```

Liblets are **Unit Tested**

```
struct IMockIDocument : public IDocument
{
    virtual ~IMockIDocument() = default;

    ::Mso::MockFunctorThrow<bool ()> mock_IsOpen;
    virtual bool IsOpen() override
    { return mock_IsOpen(); }

    struct SetDirtyArgs { bool dirty; };
    ::Mso::MockFunctorThrow<void (bool), SetDirtyArgs> mock_SetDirty;
    virtual void SetDirty(bool dirty) override
    { mock_SetDirty(dirty); }
};
```

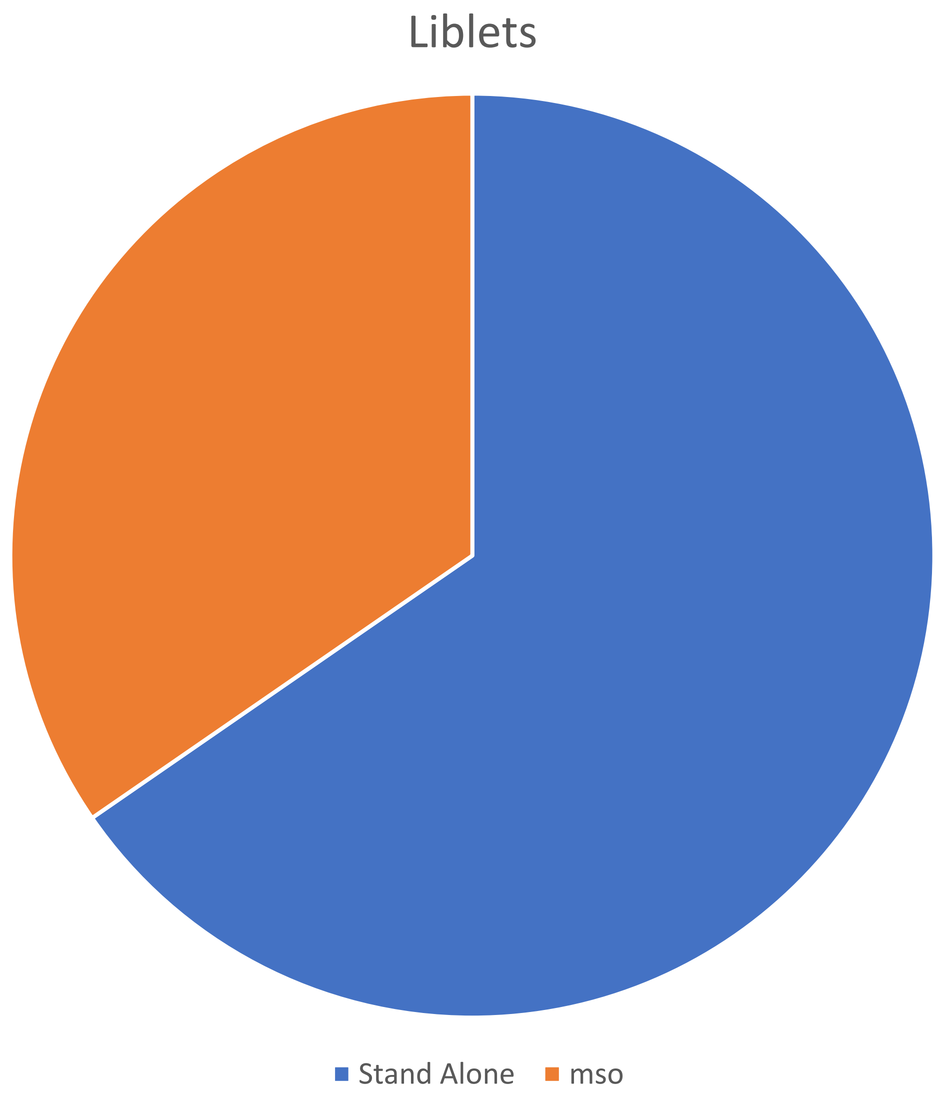
Liblet success

Liblet success

- 73% of Office projects define a liblet

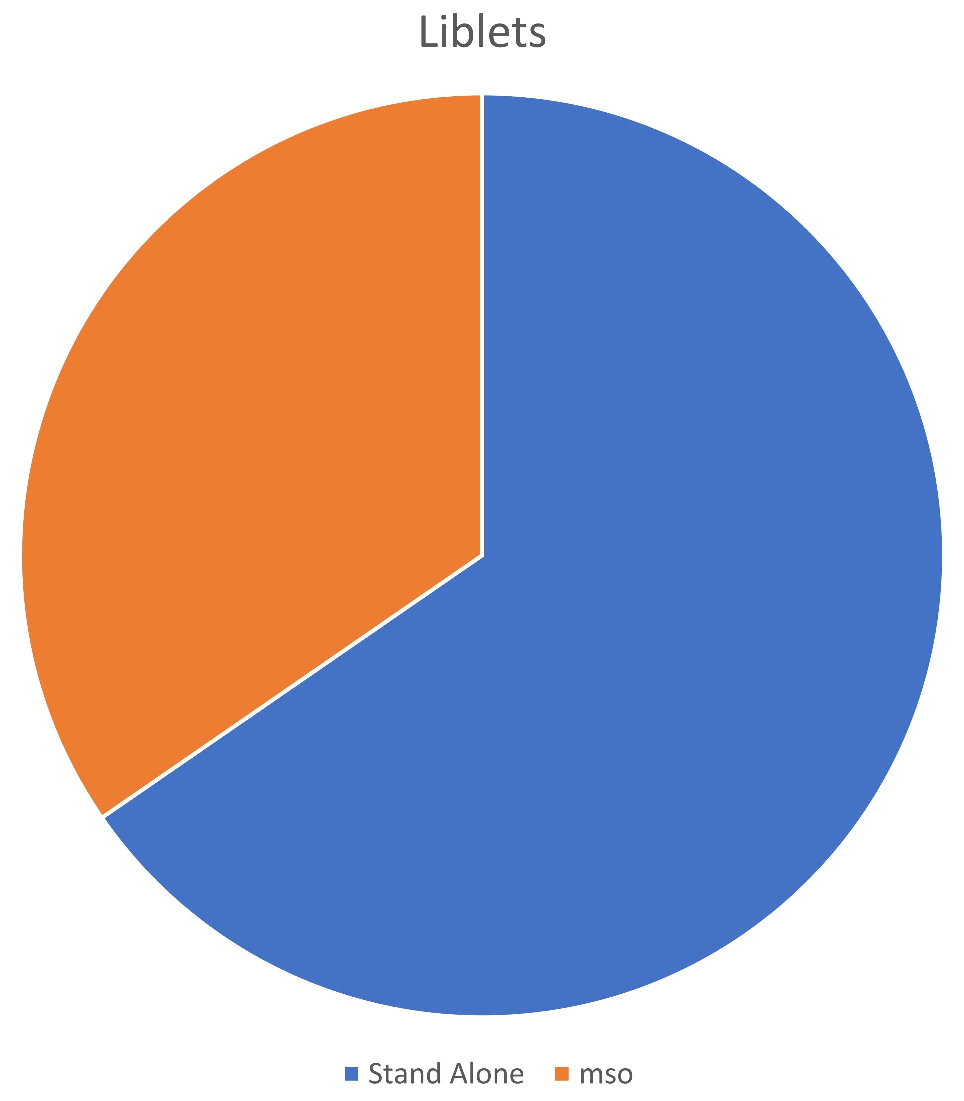
Liblet success

- 73% of Office projects define a liblet
- MSO uses liblets
    - Dependency cycles allowed internally



Liblet success

- 73% of Office projects define a liblet
- MSO uses liblets
    - Dependency cycles allowed internally
- Clients architected as liblets



What’s next

What’s next

Header Units

What are Header Units

- Binary representation of a header file
- Produces the same format as named modules
- Recommended alternative to PCH
  - Easier to setup and use
  - Smaller on disk
  - Similar performance benefits
  - More flexible than a shared PCH

Liblets + Header Units = ♥

Liblets + Header Units = 🖤

- Self contained headers

Liblets + Header Units = ❤

- Self contained headers
- Well defined, acyclic dependencies

Liblets + Header Units = ♥

- Self contained headers
- Well defined, acyclic dependencies
- No conditional compilation!

Liblets + Header Units =

- Self contained headers
- Well defined, acyclic dependencies
- No conditional compilation!

```
#if defined(Assert)
#define ASSUME( condition ) Assert( condition )
#else
#define ASSUME( condition ) __noop()
#endif
```

Header Units

- Progress to date:
  - Created 90 header units
  - Successfully built all three mso20 dlls
  - Consumed 40% of the generated header units during the build
- Next steps
  - Performance metrics.
  - Cost vs benefits on header unit “flavors”.
- Read more at https://aka.ms/officeheaderunits

How Microsoft Uses C++ to Deliver Office

Huge Size, Small Components

ZACHARY HENKEL

Cppcon
The C++ Conference

2022
September 12th-16th

Thank You

Enjoy the rest of the conference!

Join #visual_studio channel on CppCon Discord
https://aka.ms/cppcon/discord

- Meet the Microsoft C++ team
- Ask any questions
- Discuss the latest announcements

Take our survey
https://aka.ms/cppcon

Our sessions

Monday 12th

- GitHub Features Every C++ Developer Should Know – Michael Price
- The Imperatives Must Go – Victor Ciura
- What's New in C++ 23 – Sy Brand
- C++ Dependencies Don't Have to Be Painful – Augustin Popa
- How Microsoft Uses C++ to Deliver Office – Zachary Henkel

Tuesday 13th

- High-performance Load-time Implementation Selection – Joe Bialek, Pranav Kant
- C++ MythBusters – Victor Ciura

Wednesday 14th

- -memory-safe C++ - Jim Radigan

Thursday 15th

- What's New for You in Visual Studio Code – Marian Luparu, Sinem Akinci
- Overcoming Embedded Development Tooling Challenges – Marc Goodner
- Reproducible Developer Environments – Michael Price

Friday 16th

- What's New in Visual Studio 2022 – Marian Luparu, Sy Brand
- C++ Complexity (Keynote) – Herb Sutter

Resources

- CppCon 2014: Zaika Antoun "Microsoft w/ C++ to Deliver Office Across Different Platforms, Part I" – YouTube
- CppCon 2014: Zaika Antoun "Microsoft w/ C++ to Deliver Office Across Different Platforms, Part II" – YouTube
- Walkthrough: Build and import header units in Visual C++ projects | Microsoft Docs
- Microsoft C++ Team Blog