01/02/2013

Autodesk Autoloader White Paper

by Fenton Webb

Here's a draft White Paper for the Autodesk Autoloader, please feel free to comment but before you do please consider these points:

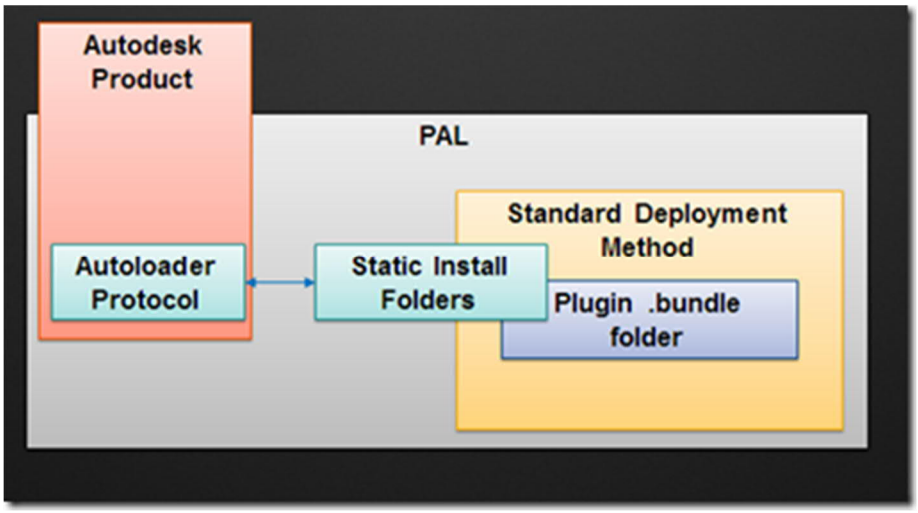
1) The Autoloader is designed to simplify 90% of all Autodesk application deployments and is built on-top-of any existing plugin architectures. If the Autoloader doesn't do what you need, feel free to use the old style of deployment. With that said, if the Autoloader doesn't do what you want, please let ADN know so that the Autoloader team can consider your Use case.

2) The Autoloader is still being enhanced, although the AutoCAD implementation is currently nearing completion. The AutoCAD team implemented the first version of the Autoloader with the view of it being adopted by the other Autodesk Product teams – Revit, Inventor, Alias, Vault, 3DS Max, Maya, now all implement the Autoloader plugin architecture.

3) Check out this PPT, it was presented at Autodesk Universityin 2010, it details an AutoCAD specific overview of why we invented the Autoloader.

Autodesk Autoloader Overview

The "Autoloader" module is intended to be a universal plugin loading mechanism for all Autodesk products. Exactly how the "Autoloader" works for an individual product may vary depending on the Autodesk product's needs, however, the basic design and implementation will be the same.



The "Autoloader" plugin mechanism simplifies deployment of your plugin applications. This is done by allowing you to deploy your plugins as a simple package format (a folder structure with a .bundle extension) along with an XML file placed in the root of the folder structure. The XML file contains metadata which describes the components of your plugin inside the folder structure, and how they should be loaded. The metadata XML also contains information about which Autodesk products it runs with, Operating Systems it supports (Mac, Win32, Win64, Web and/or Mobile) and also Autodesk version/series and/or verticals that it runs with.

The AutoCAD implementation of the "Autoloader" not only supports the loading of plugin runtime modules such as ARX, DBX, LSP, FAS, VLX etc but it also supports the loading of other AutoCAD modules types such as Partial CUIX files, ToolPalettes and more.

Other Autodesk Products utilize the same metadata XML, loaded from the same locations as AutoCAD apps, to describe how to load their own module types. This allows single Apps to define multiple cross product plugins.

The package (or rather bundle folder structure) is "installed" simply by placing a copy of the bundle in one of the static Autodesk Application Plugin folders on your hard drive, depending on your machine installation requirement, user and/or administrative requirements. The design requires that no registration is done by an installer at all - all App registration is automatically carried out by the host Autodesk Product "Autoloader" component and/or by the loading modules themselves.

By moving the registration away from the App's installer to the host product's "Autoloader":

1) Autodesk App installers become much easier to create
2) Each individual Autodesk product App deployment requirements become uniform, standardized and work the same even on a Mac machine (cross platform)
3) Individual Autodesk product App registration complexity is removed.

The folder locations are:

- %APPDATA%/Autodesk/ApplicationPlugins
  - Requires lowest User administrative rights.
  - Will not install for all users, only for the current user.
- %ProgramFiles%/Autodesk/ApplicationPlugins (AutoCAD 2012/deprecated in 2013)
  %ProgramData%/Autodesk/ApplicationPlugins (AutoCAD 2013)
  - Requires administrative rights to install to as per Windows Vista/7 requirements.
  - However, the need for a secondary installer is removed because the path is not user specific.

NOTE: To test where %APPDATA% or %ProgramFiles% or %ProgramData% resides on your machine, simply enter the string %APPDATA% or %ProgramFiles% or %ProgramData% into your Windows Explorer address bar.

The folders previously listed are static, in other words, they are not configurable and cannot be changed. This design ensures that deployment using installer technologies such as Microsoft's MSI or Apple's PKG is extremely simple (uses a very simple "CopyTo" protocol), and it also ensures similar deployment methods on other platforms making it fully cross platform.

Also, plugin management tools such as the Autodesk AppManager can very easily control all Autodesk Apps installed on the system because the runtime file reside in the same place. In addition to that, Product Support is also made easier for the same reason.

NOTE: You may manually copy the package .bundle to one of the folders if required, it's not necessary to create an installer. However, a professional deployment should have an installer built so that version upgrades and repairs can be done in an easy way.

NOTE: A single package .bundle can be deployed on multiple operating systems if required. Win32 and Win64 are required as minimum OS support for a bundle.

All Autodesk "Autoloader" implementations check the static folders in the order they were previously listed, top to bottom, in two specific contexts

- Autodesk Product Startup
- While the Autodesk Product is running, and an Application 'Appears' in either of the two folders.
    - **NOTE:** When the running Autodesk Product detects a new application in either of the two static folders, it waits six seconds for any file copying to finish before processing the package. If the installation of the application program files takes more than six seconds it is recommended that you advise your users to close the Autodesk Product before installing.

When the Autodesk Product finds an application package bundle that hasn't been processed yet, it opens the XML metadata file describing the App and automatically registers and/or (if required) loads, all of the compatible modules contained within.

For Autodesk Exchange downloads, deployment of the package is done via a simple MSI installer. For early applications, we ask that you provide Autodesk with the package contents and XML metadata file, and we will build an MSI installer for you using the Autodesk AppPacker tool (currently internal only).

Package format and contents

There is a lot of detail in this document intended to cover some of the more complex applications you might deploy, but the package can be very simple for a basic plugin application. The package format does have a set layout which is interpreted by the product Autoloader, however, there is currently no set schema therefore you can add your own App specific XML settings data to the package meta data, if required.

Here is an example of one of the simplest package formats and accompanying XML metadata file (blue text indicates a folder, black text is a file, explanations in italic):

ExamplePlugin1.bundle – *The folder containing your plugin will have an extension of .bundle*

- PackageContents.xml – *The XML metadata file used to describe the files in the bundle*
- Minimal.lsp – *An AutoCAD AutoLISP file*
- Minimal.cuix – *An AutoCAD CUIX file*
- FentiCAD.ico – *An icon used by the Apps store (the file type used can also be .bmp)*
- Help.html – *Either the main help file itself or a document referring the user to more detailed help (e.g. online help at your company's website)*

A simple XML metadata (the PackageContents.xml file) describing an AutoLISP Application which loads on Startup and for each document opened in the session (PerDocument=True).

```
<?xmlversion="1.0"encoding="utf-8"?>
<ApplicationPackageSchemaVersion="1.0"
        AutodeskProduct="AutoCAD"
        Name="Minimal LISP"
        Description="An AutoCAD LISP Only program"
        AppVersion="1.0.0"
        HelpFile="./Contents/Resources/Help.html"
        Author="Fenton Webb"
        Icon="./Contents/Resources/FentiCAD.ico">

 <CompanyDetailsName="FentiCAD"Url="www.FentiCAD.com"Email="fenton.webb@autodesk.com" />
 <ComponentsDescription="Runtime parts">
  <RuntimeRequirementsOS="Win32|Win64|MacOS"Platform="AutoCAD|AutoCAD*"SeriesMin="R18.2"SeriesMax="R19
.1" />
  <ComponentEntryAppName="MinimalLISP"Version="1.0.0"ModuleName="./Contents/Runtime/Minimal.cuix" />
```

```
  <ComponentEntryAppName="MinimalLISP"Version="1.0.0"ModuleName="./Contents/Runtime/Minimal.lsp"PerDocument="True"/>
 </Components>
</ApplicationPackage>
```

Note: Notice that all path specifiers are "/" not "\", and paths are relative to the root .bundle folder. The bundle *must* be self contained where at all possible – none self contained bundles break cross platform compatibility.

The package format is simply a folder structure with the root folder having the .bundle extension. You can create whatever subfolder structure you like but we recommend the following folder structure where possible so that the tools the AppStore team uses to build your app installer can follow a standard style.

The XML metadata file must be called **PackageContents.xml** and must also reside in the root folder.

Below is an example plugin folder structure where the App has no OS dependant modules and is also very small (folders are in blue and files are in black)

ExamplePlugin2.bundle
- PackageContents.xml
- Contents
  - mainCommands.lsp
  - test.lsp
  - Resources
    - MypartialCUI.cuix
    - Icon1.ico
  - Help
    - HelpHome.htm

As an example, you can use whatever structure you like, even just placing all files in the root folder. The above could also have been packaged like this:

ExamplePlugin2.bundle
- PackageContents.xml
- mainCommands.lsp
- test.lsp
- MypartialCUI.cuix
- Icon1.ico
- HelpHome.htm

NOTE: However, combining all files into one folder can make it harder to tell what is in the folder and for larger applications it can also lead to load time performance issues, so it is not recommended.

For ObjectARX applications, where the specific runtime is important, we suggest that you use OS specific folders such as Mac (for Apple Mac), Win32 (for Windows 32-bit) and Win64 (for Windows 64-bit), like this:

- ExamplePlugin3.bundle
  - PackageContents.xml
  - Contents
    - Win32
      - MyArx_win32.arx
    - Win64
      - MyArx_x64.arx
    - Mac
      - MyArx.bundle
    - CrossPlatform
      - mainCommands.lsp
      - test.lsp
    - Resources
      - MypartialCUI.cuix
      - Icon1.ico
    - Help
      - HelpHome.htm

If you have multi-version support, then you should include that in version specific folders

- ExamplePlugin3.bundle
  - PackageContents.xml
  - Contents
    - Win32
      - 2013
        - MyArx_win32.arx
      - 2014
        - MyArx_win32.arx
    - Win64
      - 2013
        - MyArx_x64.arx
      - 2014
        - MyArx_x64.arx
    - Mac
      - 2013
        - MyArx.bundle
      - 2014
        - MyArx.bundle

CrossPlatform

mainCommands.lsp

test.lsp

Resources

MypartialCUI.cuix

Icon1.ico

Help

HelpHome.htm

If you have multi-version, multi-product support in the same bundle, then you should structure your bundle folders like this…

ExamplePlugin3.bundle

PackageContents.xml

Contents

Win32

AutoCAD

2013

MyArx_win32.arx

2014

MyArx_win32.arx

Revit

2013

MyRevit.addin

2014

MyRevit.addin

Etc…

Additional package examples accompany the example packages later in this document. Once you have completed creating your own .bundle folder structure containing your application and a PackageContents.xml (in the root of the .bundle folder) please send it to us to be put into our standard installation MSI.

PackageContents.xml format

The PackageContents.xml file contains some basic information about you (the plugin developer) and the application, and specific information about your plugin components – such as load behaviour (load on startup, command invocation, etc), operating systems, and the AutoCAD and vertical releases it is compatible with.

The following is an example of including basic information in the XML metadata file, followed by a description of the XML elements. You can see more examples in the AutoCAD Customization Guide.

```
<?xml version="1.0" encoding="utf-8"?>

<ApplicationPackage SchemaVersion="1.0" ProductType="Application"
                    Name="SampPackage"
                    AppVersion="1.0.0"
```

```
                Description="Sample packageXml"
                Author="Fenton Webb"
                Icon="./Contents/Resources/ProductIcon.ico"
                HelpFile="./Contents/Resources/HelpFile.htm"
                ProductCode="{9BD0898D-3B74-4FB0-835A-8D64040A307E}"
        UpgradeCode="{634B7497-F903-4DA8-8099-F9D8D2D88650}">

 <-- overview runtime requirements of the entire package, for start-up performance and easy view information -->
 <RuntimeRequirements OS="Win32|Win64|Mac" Platform="AutoCAD|Civil3D"
        SeriesMin="R18" SeriesMax="R18.2" />

<CompanyDetails
        Name="Fent Soft" Phone="415 777 8888"
        Url="www.fentsoft.com" Email="fenton.webb@autodesk.com" />

  <Components>

   <RuntimeRequirements SupportPath="./Contents/Support"
        OS="Win32|Win64" Platform="AutoCAD|Civil3D"
        SeriesMin="R18" SeriesMax="R18.1" />

   <ComponentEntry AppName="MyApplication"
     ModuleNameWin32="./Contents/Win32/MyApplication.arx"
     ModuleNameWin64="./Contents/Win64/MyApplication64.arx"
                AppDescription="Windows 32 and 64bit support" LoadOnCommandInvocation="True">
     <Commands GroupName="ADSK_AUTOLOADER">
       <Command Local="_test" Global="_test" />
       <Command Local="_test2" Global="_test2" />
     </Commands>
   </ComponentEntry>
   <ComponentEntry ModuleName="./Contents/Resources/WindowsCUI.cuix" AppDescription="Windows CUI File" />
  </Components>

  <Components>
   <RuntimeRequirements SupportPath="./Contents/Support" OS="Mac" Platform="AutoCAD|Civil3D" SeriesMin="R18.0"
SeriesMax="R18.2" />
   <ComponentEntry AppName="MyArxApp" ModuleName="default.bundle" AppDescription="A Mac native bundle entry"
LoadOnCommandInvocation="True">
     <Commands GroupName="ADSK_AUTOLOADER">
       <Command Local="_test" Global="_test" />
       <Command Local="_test2" Global="_test2" />
     </Commands>
   </ComponentEntry>
   <ComponentEntry AppName="MyLispApp" ModuleName="./Contents/Resources/test.lsp" AppDescription="A very simple lisp
file" PerDocument="False" />
   <ComponentEntry ModuleName="./Contents/Resources/MacCUI.cuix" AppDescription="Mac Specific CUI File" />
  </Components>

</ApplicationPackage>
```

Definitions of element:

NOTE: XML attributes are case sensitive

Element name: ApplicationPackage (required)

You can specify RuntimeRequirements XML elements and Components XML elements within the ApplicationPackage element. RuntimeRequirements specified at the ApplicationPackage level are intended to give an overview of the entire bundle but they may or may not exist depending on the application. Defining the Application RuntimeRequirement section is recommended as it provides faster processing by the product specific Autoloader modules, provides a quicker/easier way for external applications to understand the intended product focus of the App and also provides information about the intended product focus for Apps which have no Components sections (one's without Autoloader loadable/processable modules).

Defines high level information about the application:

Attributes:

SchemaVersion (required): Version number for PackageContents.xml format. Should always be 1.0 until a newer format is released.

AutodeskProduct (required): This field gives an overview of the bundle's intended Autodesk product. An example would be, you might have a Kitchen application that has been developed on AutoCAD so it is an AutoCAD product, however, you may have data extensions or importers into other products like Revit and Inventor that need to be included in the bundle. Those extensions will of course be loaded by the Revit and Inventor Autoloader, but that does not make your Kitchen application a Revit or Inventor app, it is an AutoCAD app.

ProductType (required): This field is used by the AppStore installer team to indicate an "Application" or "Content" installer.

AppVersion (required): Version number for your plug-in application. AutoCAD uses this to determine if an updated version of your application is available and then informs your customer.

NOTE: In order for the UpgradeCode system to work, you must increment the AppVersion number. It is recommended that the AppVersion is in the format "1.0.0", or "1.0.0.0" depending on your requirements.

Author (optional): Name of the author of the plug-in bundle.

Name (required): Name of your plug-in application. If you need localized names you can specify these using NameDeu, NameEsp, NameFra, etc (see full list later in document).

Description (required): Short (one paragraph) description of your plug-in application. Localized descriptions can be specified (see later in document).

Icon (required): Icon to be used to represent your plug-in application in the Apps store. Icon size should be 32x32 pixels and up to 32-bit (truecolor) color depth, we recommend BMP or ICO.

Helpfile (required): Location of the file within the package, HTML or PDF. The file can be either the full help or contain information on how to access help.

SupportedLocales (optional): this attribute is used by the Autodesk AppPacker tool, it is used to indicate the language types that are supported by the bundle. When this attribute is included, the AppPacker tool reads the delimited string and adds each language locale to the Exchange Store installer MSI.

AppNameSpace (optional) : This is an additional app namespace string, it is currently only used by the Autodesk AppManager to indicate Autodesk specific products.

OnlineDocumentation (optional) : This is a Uri pointing to any extra documentation that the app utilizes.

ProductCode (required): GUID string used in two ways. First, it is used as an Add/Remove Programs installer ID and secondly it is used by AutoCAD in order to record which applications have been "Bubble Notified" on load. The ProductCode should be updated if the AppVersion changes, this is so upgrade installs will work properly and also so a fresh "Bubble" notification will be displayed for the upgrade when loaded into AutoCAD the first time.

NOTE: If you deploy your plug-in application bundle through the Autodesk Exchange, the ProductCode will automatically be assigned to your product.

UpgradeCode (optional): GUID string which must never be changed. It is a unique identifier used exclusively by the Autodesk Exchange installer. Without this code, upgrade installs will require that the old version be uninstalled, and give a bad user experience. If the code is maintained, the user will install and upgrade with ease.

NOTE: In order for the UpgradeCode system to work, you must increment the AppVersion number. It is recommended that the AppVersion is in the format "1.0.0", "1.0.0.0" depending on your requirements.

Element name: CompanyDetails (required)

Information about your company.

Attributes:

Name (required): Your name or company name.

Phone (optional): Your company phone number, localized phone numbers can be specified (see later in document).

Url (required): Your company or product website, localized URLs can be specified (see later in document) – a URL is required for the Autodesk AppStore Digital Signature to be applied to the MSI installer.

Email (required): Contact email address, localized email addresses can be specified (see later in document).

Element name**: Components** (required)

You can specify RuntimeRequirements XML elements and ComponentEntry XML elements within the Components element. RuntimeRequirements specified at the Components level controls whether that Components section is processed by the Autoloader, but they may or may not exist depending on the application.

Contains information about a set of components that make up one part of your application. For example:

1. A Components section might hold the 32-bit version of your app, another the 64-bit version

2. A different Components section might hold your AutoCAD 2011 version and another your 2012 version (you can specify specific RuntimeRequirements element inside of the Components->ComponentEntry element)
3. A different Components section may contain plugins to other products, for instance, a data import tool for Revit and another for Inventor

Information on platform, product, and version/series information is defined in the RuntimeRequirements element properties.

The individual modules specified by ComponentEntry XML elements (see below) within this section are loaded from bottom to top. Therefore, any component on which another component has a dependency must be lower down the list. For example, if an ObjectARX module is dependent on an ObjectDBX module, then the ObjectARX module will appear above the ObjectDBX module in the list.

Different Components elements should not have dependencies on each other. For example, if you have a LSP file that is OS/version independent but depends on an ARX file that is OS/version specific, then you should include the LSP file in each Components element that references the different ARX modules. However, if your LSP file is not dependent on any other files, then it can be broken out into its own (platform/version independent Components element).

Attributes:

> **Description** (optional): Description of the Components section, this is purely used by the Autodesk AppPacker UI.

Element name: RuntimeRequirements (optional)

Optionally defined as a child elelment of the ApplicationPackage, Components and/or ComponentEntry elements. This element defines details about the different products types, product versions, platforms, and languages that are supported at the element level that it's defined.

You should always include at least an ApplicationPackage RuntimeRequirements, otherwise your bundle will be blindly processed by multiple Autodesk products, potentially incompatible hosts.

If you do not specify an ApplicationPackage level RuntimeRequirements element, then you must define a RuntimeRequirements element for each Components section you define. Your Components sections RuntimeRequirements element may define multiple Platforms, OSs and series values as long as you define each ComponentEntry with a specific focused RuntimeRequirements section to suit.

Attributes:

> **OS** (optional): Values are ‘Mac’, ‘Win32’, and ‘Win64’. If omitted, it is assumed your application will run on all operating systems – which would only be true for LSP applications and CUIx files, for instance.
>
> **Platform** (optional): Indicates the products that the plug-in can be loaded into. Used to specify if the section can be loaded into vanilla AutoCAD or a vertical AutoCAD and can be set to one or more of the following abbreviated strings:
>
> Civil3D - Autodesk Civil 3D
> AutoCAD - AutoCAD (AutoCAD* indicates all AutoCAD based products – available from AutoCAD 2012 sp1)
> Map - Map
> AIS - Inventor Series (AIS)
> ADT - Architectural Desktop

ACADM - AutoCAD Mechanical (ACADM)
MEP - AutoCAD MEP
ACADE - AutoCAD Electrical (ACADE)
LDT - Land Desktop
AIP - Inventor Professional (AIP)
AIPRS - Inventor Professional for Routed Systems (AIPRS)
AIPSIM - Inventor Professional for Simulation (AIPSIM)
PNID - AutoCAD P & ID - 2D
Plant3D - AutoCAD Plant 3D
Civil - Autodesk Civil

Other Platform string names (that may or may not be in use currently) are:

ecscad
Simulation Mechanical
Simulation Moldflow Adviser
Simulation Moldflow CAD
Simulation Moldflow Design
Simulation Moldflow Insight
Simulation Moldflow Synergy
Simulation Moldflow Communicator
Simulation Multiphysics
Simulation CFD
Simulation CFD Advanced
Simulation CFD Motion
Alias
Alias*
Alias Design
Alias Surface
Alias Automotive
Inventor
Inventor Publisher
Inventor Factory
Revit
Revit Architecture
Revit Structure
Revit MEP
Vault

Here is an example Platform entry with all of the AutoCAD flavours supported:
Platform="AutoCAD*|AutoCAD"

Here is an example Platform entry with just the verticals
Platform="Civil3D|Map|ADT|ACADM|MEP|ACADE|LDT|PNID|Plant3D|Civil|AIS|AIP|AIPRS|AIPSIM"

If not included, it is assumed your plugin can be loaded into all AutoCAD and all its vertical applications.

You may specify multiple products by using the '|' symbol as a separator, for instance Platform="AutoCAD|Civil3D".

AutoCAD 2012 and AutoCAD 2012-based products currently do not support this attribute.

SeriesMin (optional): Defines the minimum release number for 'this' set of components. Can be a major release number only (e.g. AutoCAD R18), or specific version (e.g. AutoCAD R18.0). Each Autodesk product has a unique series string. As an example – AutoCAD – R18.0 is the same as the release number in the AutoCAD registry hive. If this and SeriesMax are not specified, then it is assumed these components are compatible with all of the releases, e.g. a LISP file in AutoCAD. If you omit this value than any release lower than SeriesMax is allowed. All none AutoCAD products use their year of the release with a single character prefix depicting the product e.g. R2013 is Revit 2013, V2013 is Vault etc.

SeriesMax (optional): Defines the maximum release number for this set of components. Specified the same as SeriesMin. If you omit this value then any release above **SeriesMin** is allowed. All none AutoCAD products use their year of the release with a single character prefix depicting the product e.g. R2013 is Revit 2013, V2013 is Vault, Ir18 is Inventor 2013 etc.

SupportPath (optional): Semicolon separated list of support paths used by this set of components (this is normally a relative path to a location within the plug-in application bundle). You may specify a localized version of this attribute if required. This path is added to the product's Support path – for AutoCAD, the 'Autoloader' adds this path to the Files->Support path Options.

ToolPalettePath (optional): Semicolon separated list of support paths used by this set of components (this is normally a relative path to a location within the plug-in application bundle). You may specify a localized version of this attribute if required. This path is added to the product's Toolpalette path.

Element name: ComponentEntry (required)

You can specify RuntimeRequirements XML elements and Commands XML elements within the ComponentEntry element. RuntimeRequirements specified at the ComponentEntry level controls whether that ComponentEntry section is processed by the Autoloader, but they may or may not exist depending on the application.

This element is contained within the Components element. It describes specific details about an individual component/module within the Components element. You can specify as many ComponentEntry's as you like and it may also contain a Commands element if LoadOnCommandInvocation is specified. Component types can be ObjectARX, ObjectDBX, AutoLISP, .NET or partial CUIX.

IMPORTANT: It is strongly recommended to use LoadOnCommandInvocation if your application defines commands. This is to ensure that AutoCAD startup times are kept as small as possible.

NOTE: (For AutoCAD ARX, DBX and .NET developers only) If the number of ComponentEntry's in a same Components element is greater than one and at least one ComponentEntry is an ARX, DBX or .NET DLL, then the path defined in the ModuleName attribute of that ComponentEntry will be added to the system PATH. This is so file dependencies are automatically resolved, and any dangers for missing DLL's are handled as best as possible.

NOTE: LoadFrom() context issues are automatically handled by the Autodesk product 'Autoloader'. .NET DLL's that used to require loading from the host Autodesk product exe folder can now be loaded from folders away from the exe folder. The Autodesk product 'Autoloader' implements the AssemblyResolve event which automatically handles this.

NOTE: For AutoCAD users, partial CUI files that implement WPF custom controls can be declared using the RibbonControls/ RibbonControl and AssemblyMappings/AssemblyMapping elements.

Attributes:

AppName (Required): Name of this component. For AutoCAD users, this is the same as AcadAppInfo class AppName in ObjectARX API and .NET.

AppDescription (optional): A brief description about this particular component entry. For AutoCAD users, this is the same as AcadAppInfo class AppDescription in ObjectARX API and .NET.

Version (recommended): Version number associated with the ComponentEntry.

ModuleName (required): Relative path to component within the bundle. For AutoCAD users, this is the same as AcadAppInfo class ModuleName in ObjectARX API, but it also supports AutoLISP modules also.

The component type is inferred from the file extension – e.g. for AutoCAD users, 'arx' = ObjectARX, 'dll' = .NET, 'lsp' = AutoLISP, etc. However, you may override this inferred type selection by using the AppType parameter. It is expected that your application will handle multiple languages internally (for example, through the use of resource DLLs). If it is absolutely necessary to ship different versions of a specific component to support different languages, then you can add a suffix to ModuleName attribute to specify multiple languages. (E.g. To support English and French, specify two additional parameters: ModuleNameEnu and ModuleNameFra.)

AppType (optional): The applications type is automatically inferred from the ModuleName attribute's file extension, however, you may override the inferred application type by using this optional attribute.

The AutoCAD Autoloader currently processes and recognizes these settings:

"Bundle","ARX","Lisp","CompiledLisp","Dbx",".NET","Cui","CuiX","Mnu" and "Dependency"

"Dependency" is used where you have a module that should NOT be processed by AutoCAD. An example would be say a licensing DLL, or maybe a resource DLL.

PerDocument (optional): Controls the loading of a module when a document is loaded into the editor. For AutoCAD users, if a LSP file should be loaded per document if present, default is True.

***'LoadReasons'* see below for actual attribute name** (optional and multiple): Defines the loading flags for a particular module. For AutoCAD users, see the ObjectARX reference for AcadAppInfo LoadReasons for full details. Set parameter value to "**True**" to enable a load behavior and "**False**" to disable it. With the exception of *LoadOnCommandInvocation*, all LoadReasons are enabled by default if not specified. You must explicitly include the LoadReason and set its value to False to disable it. *LoadOnCommandInvocation* is a special case – it is disabled by default, and enabling it will disable all other LoadReasons by default (unless they are explicitly enable).

> ***LoadOnCommandInvocation*** – Load only when one of your custom commands is invoked. When using this Load Reason, you must include a 'Commands' element. Also, if *LoadOnCommandInvocation* is enabled, then LoadReasons *LoadOnAutoCADStartup* and *LoadOnAppearance* are assumed to be disabled unless explicitly enabled. Only applies to ARX and .NET modules.
>
> ***LoadOnAutoCADStartup*** – Load when AutoCAD starts up. If this is specified, then it has precedence over all other LoadReasons. We recommend only using *LoadOnAutoCADStartup* when none of the other load reasons are suitable – i.e. you should explicitly disable it (set to False) whenever possible. If the *LoadOnAutoCADStartup* parameter is omitted, then it defaults to True; unless *LoadOnCommandInvocation* is True, in which case *LoadOnAutoCADStartup* defaults to False.

LoadOnProxyDetection – Load when a proxy for your custom entity is detected. The default is to enable this (i.e. =True) unless it is explicitly set to False (disabled). When enabling *LoadOnProxyDetection*, *LoadOnAutoCADStartup* should be set to False. Only applies to DBX modules.

LoadOnAppearance - Load the module as soon as AutoCAD detects the application bundle in its ApplicationPlugins folder, thereby supporting instant load on installation with no need to restart AutoCAD. The property defaults behave the same way as ***LoadOnAutoCADStartup***, except that the load context is relevant only when an application is installed while AutoCAD is running.

Element name: Commands (required if LoadReason *LoadOnCommandInvocation* is specified)

You can specify Command XML elements within the Commands element.

This element specifies command names that should be registered for LoadOnCommandInvocation. It can contain one or more Command elements.

NOTE: Defining LoadOnCommandInvocation=True automatically turns off LoadOnAutoCADStartup and LoadOnAppearance.

Attributes:

GroupName (Required): Group name that a command is associated with. This is used in the registry in order to record their existence for inline Command auto complete and for command invocation demand loading.

You must prefix the GroupName string with your Registered Developer Symbol text so as to avoid possible name clashing with other products. For AutoCAD users, see the ObjectARX AcadAppInfo documentation for more help.

Element name: Command (required if Commands element is present)

Specifies the global and local names for each command.

Attributes:

Global (Required): Global name of the command. You must prefix your global commands with your Registered Developer Symbol text so as to avoid possible name clashing with other products. For AutoCAD users, see the ObjectARX AcadAppInfo documentation for more help.

Local (Required): Local name of the command. It is highly recommended that you prefix your local commands with your Registered Developer Symbol text so as to avoid possible name clashing with other products. For AutoCAD users, see the ObjectARX AcadAppInfo documentation for more help. If Multilanguage support is required, the Local attribute can be replaced by LocalEnu, LocalDeu, LocalFra, etc… to specify names for each locale.

Description (Required): A single sentence describing what the command does.

HelpTopic: (Optional) an HTML anchor reference to the ApplicationPackage::HelpFile above. This help topic is invoked when your command is running and F1 is pressed.

StartupCommand: (Optional) set to “True” or “False” – if true, then the command will be invoked at startup

Localized attributes

Some attributes mentioned earlier in the document can also be localized by specifying the attribute name and a three letter suffix. The Autoloader checks the running language, if the attribute matches the running language then the localized attribute is used, otherwise the non-localized version of the attribute is used – which should be English.

So, for example, the 'Local' attribute of the 'Command' element can be localized as follows:

LocalEnu = English

LocalDeu = German

LocalFra = French

LocalEsp = Spanish

Etc.

Or – the following example shows that German and Spanish are localized, all other language versions use HelpFile (English)

HelpFile="./Contents/Resources/Help.html"

HelpFileDeu="./Contents/Resources/HelpDeu.html"

HelpFileEsp="./Contents/Resources/HelpEsp.html"

We recommend you always include a generic (non-localized) parameter to be used in case your plug-in is loaded in a host Autodesk product version you haven't provided localized information for.

The locale codes for shipping Autodesk releases are as follows:

Locale code	**Description**
Chs	Chinese (PRC)
Cht	Chinese (Taiwan)
Csy	Czech
Deu	German
Enu	English
Esp	Spanish
Fra	French
Hun	Hungarian
Ita	Italian
Jpn	Japanese
Kor	Korean
Plk	Polish
Rus	Russian

Example packages

The best place to find example packages is on the Autodesk Exchange store, that said, here are some to look at…

A LSP files bundle, load one file for each new document and the other just on startup, with a CUIX file. No version, language, or platform version is specified; this is assumed to be compatible with all.

```
<?xmlversion="1.0"encoding="utf-8"?>
<ApplicationPackageSchemaVersion="1.0"ProductType="Application"
                Name="Simple LISP"
                AppVersion="1.0.0"
                Description="An AutoLISP Application containing 2 LSP files"
                Author="Fenton Webb"
                Icon="./Contents/Resources/icon1.ico"
                HelpFile="./Contents/Help/HelpFile.htm">

 <RuntimeRequirementsPlatform="AutoCAD*"/>
 <CompanyDetailsName="Fent Soft"Phone="415 777 8888"
           Url="www.fentsoft.com"Email="fenton.webb@autodesk.com" />
 <Components>
  <RuntimeRequirementsSupportPath="./Contents/Support" />
  <ComponentEntryAppName="MainLISP"
          ModuleName="./Contents/mainCommands.lsp"PerDocument="True" />
  <ComponentEntryAppName="2nd LISP"ModuleName="./Contents/test.lsp"PerDocument="False" />
  <ComponentEntryModuleName="./Contents/Resources/MypartialCui.cuix" />
 </Components>
</ApplicationPackage>
```

File Structure:

Folders are in blue and files are in black.

ExamplePlugin1.bundle

PackageContents.xml

Contents

mainCommands.lsp

test.lsp

Resources

MypartialCUI.cuix

Icon1.ico

Help

HelpFile.htm

An application consisting of the ObjectARX sample ‘Polysamp’ built for the Mac, Windows 32bit and Windows 64bit – a single bundle defining the runtime for three separate computer platforms.

Notice that the ComponentEntry elements are in reverse dependency order (AutoCAD loads modules from bottom to top). The Windows version would declare the corresponding Components element’s RuntimeRequirements as OS=”Win32” or OS=”Win64” or you can combine them using OS=”Win32|Win64” as seen in the example below.

The Mac ARX entry has no path specified in the ModuleName, this is because on the Mac an ARX application is already in a .bundle format and therefore the .bundle is the ARX.

Please note the AppDescription for each ComponentEntry, it describes what each represents.

```
<?xmlversion="1.0"encoding="utf-8"?>
<ApplicationPackageSchemaVersion="1.0"ProductType="Application"ProductCode="{324994BC-1E39-49F5-9B66-
3C7549E7F636}"UpgradeCode="{224C1293-C62A-4B9C-ABB4-1C09779FB44E}"
                Name="PolySamp"
                AppVersion="1.0.0"
                Description="Sample packageXml"
                Author="Fenton Webb"
                Icon="./Contents/Resources/icon.ico"
                HelpFile="./Contents/Resources/ReadMe.htm" >

<RuntimeRequirementsOS="Mac|Win32|Win64"Platform="AutoCAD"SeriesMin="R18"SeriesMax="R18.2" />
  <CompanyDetailsName="Fent Soft"Phone="415 777 8888"Url="www.fentsoft.com"Email="fenton.webb@autodesk.com" />

  <Components>
    <RuntimeRequirementsOS="Mac"Platform="AutoCAD"SeriesMin="R18"SeriesMax="R18.2" />
    <ComponentEntryAppName="AsdkPolyCAD"ModuleName="asdkpolyui.bundle"AppDescription="Mac polysamp UI ARX,
notice that the .bundle IS the ARX so no path in the ModuleName"
                LoadOnCommandInvocation="True">
      <CommandsGroupName="ASDK_POLYGON">
        <CommandLocal="POLY"Global="ASDK_POLY" />
        <CommandLocal="PPOLY"Global="PPOLY" />
        <CommandLocal="DRAGPOLY"Global="ASDK_DRAGPOLY" />
        <CommandLocal="POLYEDIT"Global="ASDK_POLYEDIT" />
        <CommandLocal="TRANSACT"Global="ASDK_TRANSACT" />
        <CommandLocal="HILITPOLY"Global="ASDK_HILITPOLY" />
        <CommandLocal="HILITSOLID"Global="ASDK_HILITSOLID" />
        <CommandLocal="CREATEINSERT"Global="ASDK_CREATEINSERT" />
        <CommandLocal="HILITINSERT"Global="ASDK_HILITINSERT" />
        <CommandLocal="USEDRAGDATA"Global="ASDK_USEDRAGDATA" />
      </Commands>
    </ComponentEntry>
    <ComponentEntryAppName="AsdkPolyCAD_DBX"ModuleName="./Contents/MacOS/asdkpolyobj.dbx"
                AppDescription="Custom Poly DBX Object, even though this loads via proxy detection, it must also be loaded by the
ARX UI"
                LoadOnProxyDetection="True"LoadOnAutoCADStartup="False"LoadOnAppearance="False"/>
  </Components>
  <Components>
    <RuntimeRequirementsOS="Win32|Win64"Platform="AutoCAD"SeriesMin="R18"SeriesMax="R18.2" />
    <ComponentEntryAppName="AsdkPolyCAD"
                ModuleNameWin32="./Contents/Win32/asdkpolyui.arx"
                ModuleNameWin64="./Contents/Win64/asdkpolyui64.arx"
                AppDescription="Poly Sample - main UI ARX which everything links to"LoadOnCommandInvocation="True">
      <CommandsGroupName="ASDK_POLYGON">
        <CommandLocal="POLY"Global="ASDK_POLY" />
        <CommandLocal="PPOLY"Global="PPOLY" />
        <CommandLocal="DRAGPOLY"Global="ASDK_DRAGPOLY" />
        <CommandLocal="POLYEDIT"Global="ASDK_POLYEDIT" />
        <CommandLocal="TRANSACT"Global="ASDK_TRANSACT" />
        <CommandLocal="HILITPOLY"Global="ASDK_HILITPOLY" />
        <CommandLocal="HILITSOLID"Global="ASDK_HILITSOLID" />
        <CommandLocal="CREATEINSERT"Global="ASDK_CREATEINSERT" />
        <CommandLocal="HILITINSERT"Global="ASDK_HILITINSERT" />
        <CommandLocal="USEDRAGDATA"Global="ASDK_USEDRAGDATA" />
        <CommandLocal="POLYCLEANUI"Global="ASDK_POLYCLEANUI" />
      </Commands>
    </ComponentEntry>
    <ComponentEntryAppName="AsdkPolyCAD_DBX_COM"
                ModuleNameWin32="./Contents/Win32/asdkcompoly.dbx"
                ModuleNameWin64="./Contents/Win64/asdkcompoly64.dbx"AppDescription="Custom Poly Object C++ COM Object,
must be loaded by the UI ARX"AppType="Dependency"/>
    <ComponentEntryAppName="AsdkPolyCAD_MANAGED_TESTAPP"ModuleName="./Contents/Win32/mgPolyTestVB.dll"App
Description="Custom Poly Object .NET Wrapper Test App"LoadOnCommandInvocation="True">
```

```
    <Commands>
      <CommandLocal="TestCreate"Global="TestCreate" />
    </Commands>
  </ComponentEntry>
  <ComponentEntryAppName="TESTLISP"ModuleName="./Contents/Common/test.lsp"AppDescription="Test
LISP"PerDocument="True"/>
  <ComponentEntryAppName="AsdkPolyCAD_DBX_MANAGED"
            ModuleNameWin32="./Contents/Win32/asdkmgPoly.dll"
            ModuleNameWin64="./Contents/Win64/asdkmgPoly64.dll"
            AppDescription="Custom Poly Mixed Mode .NET Wrapper, must be loaded by the UI ARX"AppType="Dependency"/>
  <ComponentEntryAppName="AsdkPolyCAD_DBX"
            ModuleNameWin32="./Contents/Win32/asdkpolyobj.dbx"
            ModuleNameWin64="./Contents/Win64/asdkpolyobj64.dbx"
            AppDescription="Custom Poly DBX Object, even though this loads via proxy detection, it must also be loaded by the
ARX UI"
            LoadOnProxyDetection="True"LoadOnAutoCADStartup="False"LoadOnAppearance="False"/>
 </Components>
</ApplicationPackage>
```

You do not have to enable LoadReason LoadOnProxyDetection for DBX modules, as it is true by default but you do have to disable LoadOnAutoCADStartup or the LoadOnProxyDetection will be overridden with an automatic load.

When LoadOnCommandInvocation is enabled, LoadOnAutoCADStartup is disabled by default; so you do not have to explicitly disable it.

- asdkpolyui.bundle
  - PackageContents.xml
  - Contents
    - MacOS
      - asdkpolydb.dbx
    - Win32
      - asdkcompoly.dbx
      - asdkmgPoly.dll
      - asdkpolyobj.dbx
      - asdkpolyui.arx
      - mgPolyTestVB.dll
      - test.lsp
    - Win64
      - Asdkcompoly64.dbx
      - asdkmgPoly64.dll
      - asdkpolyobj64.dbx
      - asdkpolyui64.arx
    - Resources
      - ReadMe.htm

Icon1.ico

A mix of applications 1 and 2, just as an example

```
<?xmlversion="1.0"encoding="utf-8"?>
<ApplicationPackageSchemaVersion="1.0"ProductType="Application"
                Name="Simple LISP"
                AppVersion="1.0.0"
                Description="An AutoLISP Application containing 2 LSP files"
                Author="Fenton Webb"
                Icon="./Contents/Resources/icon1.ico"
                HelpFile="./Contents/Help/HelpFile.htm">

<RuntimeRequirementsOS="Mac|Win32|Win64"Platform="AutoCAD"SeriesMin="R18"SeriesMax="R18.2" />
 <CompanyDetailsName="Fent Soft"Phone="415 777 8888"
           Url="www.fentsoft.com"Email="fenton.webb@autodesk.com" />
 <Components>
  <RuntimeRequirementsSupportPath="./Contents/Support" />
  <ComponentEntryAppName="MainLISP"
          ModuleName="./Contents/mainCommands.lsp"PerDocument="True" />
  <ComponentEntryAppName="2nd LISP"ModuleName="./Contents/test.lsp"PerDocument="False" />
  <ComponentEntryModuleName="./Contents/Resources/MypartialCui.cuix" />
 </Components>

 <Components>
  <RuntimeRequirementsOS="Mac"Platform="AutoCAD"SeriesMin="R18"SeriesMax="R18.2" />
  <ComponentEntryAppName="AsdkPolyCAD"ModuleName="asdkpolyui.bundle"AppDescription="Mac polysamp UI ARX,
notice that the .bundle IS the ARX so no path in the ModuleName"
            LoadOnCommandInvocation="True">
    <CommandsGroupName="ASDK_POLYGON">
     <CommandLocal="POLY"Global="ASDK_POLY" />
     <CommandLocal="PPOLY"Global="PPOLY" />
     <CommandLocal="DRAGPOLY"Global="ASDK_DRAGPOLY" />
     <CommandLocal="POLYEDIT"Global="ASDK_POLYEDIT" />
     <CommandLocal="TRANSACT"Global="ASDK_TRANSACT" />
     <CommandLocal="HILITPOLY"Global="ASDK_HILITPOLY" />
     <CommandLocal="HILITSOLID"Global="ASDK_HILITSOLID" />
     <CommandLocal="CREATEINSERT"Global="ASDK_CREATEINSERT" />
     <CommandLocal="HILITINSERT"Global="ASDK_HILITINSERT" />
     <CommandLocal="USEDRAGDATA"Global="ASDK_USEDRAGDATA" />
    </Commands>
  </ComponentEntry>
  <ComponentEntryAppName="AsdkPolyCAD_DBX"ModuleName="./Contents/MacOS/asdkpolyobj.dbx"
            AppDescription="Custom Poly DBX Object, even though this loads via proxy detection, it must also be loaded by the
ARX UI"
            LoadOnProxyDetection="True"LoadOnAutoCADStartup="False"LoadOnAppearance="False"/>
 </Components>
 <Components>
  <RuntimeRequirementsOS="Win32|Win64"Platform="AutoCAD"SeriesMin="R18"SeriesMax="R18.2" />
  <ComponentEntryAppName="AsdkPolyCAD"
            ModuleNameWin32="./Contents/Win32/asdkpolyui.arx"
            ModuleNameWin64="./Contents/Win64/asdkpolyui64.arx"
            AppDescription="Poly Sample - main UI ARX which everything links to"LoadOnCommandInvocation="True">
    <CommandsGroupName="ASDK_POLYGON">
     <CommandLocal="POLY"Global="ASDK_POLY" />
     <CommandLocal="PPOLY"Global="PPOLY" />
     <CommandLocal="DRAGPOLY"Global="ASDK_DRAGPOLY" />
     <CommandLocal="POLYEDIT"Global="ASDK_POLYEDIT" />
     <CommandLocal="TRANSACT"Global="ASDK_TRANSACT" />
     <CommandLocal="HILITPOLY"Global="ASDK_HILITPOLY" />
     <CommandLocal="HILITSOLID"Global="ASDK_HILITSOLID" />
     <CommandLocal="CREATEINSERT"Global="ASDK_CREATEINSERT" />
```

```
      <CommandLocal="HILITINSERT"Global="ASDK_HILITINSERT" />
      <CommandLocal="USEDRAGDATA"Global="ASDK_USEDRAGDATA" />
      <CommandLocal="POLYCLEANUI"Global="ASDK_POLYCLEANUI" />
    </Commands>
  </ComponentEntry>
  <ComponentEntryAppName="AsdkPolyCAD_DBX_COM"
              ModuleNameWin32="./Contents/Win32/asdkcompoly.dbx"
              ModuleNameWin64="./Contents/Win64/asdkcompoly64.dbx"AppDescription="Custom Poly Object C++ COM Object,
must be loaded by the UI ARX"AppType="Dependency"/>
  <ComponentEntryAppName="AsdkPolyCAD_MANAGED_TESTAPP"ModuleName="./Contents/Win32/mgPolyTestVB.dll"App
Description="Custom Poly Object .NET Wrapper Test App"LoadOnCommandInvocation="True">
    <Commands>
      <CommandLocal="TestCreate"Global="TestCreate" />
    </Commands>
  </ComponentEntry>
  <ComponentEntryAppName="TESTLISP"ModuleName="./Contents/Common/test.lsp"AppDescription="Test
LISP"PerDocument="True"/>
  <ComponentEntryAppName="AsdkPolyCAD_DBX_MANAGED"
              ModuleNameWin32="./Contents/Win32/asdkmgPoly.dll"
              ModuleNameWin64="./Contents/Win64/asdkmgPoly64.dll"
              AppDescription="Custom Poly Mixed Mode .NET Wrapper, must be loaded by the UI ARX"AppType="Dependency"/>
  <ComponentEntryAppName="AsdkPolyCAD_DBX"
              ModuleNameWin32="./Contents/Win32/asdkpolyobj.dbx"
              ModuleNameWin64="./Contents/Win64/asdkpolyobj64.dbx"
              AppDescription="Custom Poly DBX Object, even though this loads via proxy detection, it must also be loaded by the
ARX UI"
              LoadOnProxyDetection="True"LoadOnAutoCADStartup="False"LoadOnAppearance="False"/>
 </Components>

</ApplicationPackage>
```

No other plug-in modules depend on the LSP modules, so generally speaking they can be kept in their own OS and version independent Components section. If the bundle (ARX) or DBX modules had a dependency on them (or vice versa) then they would have to be included in the Components element on which they depended.

APPAUTOLOAD AutoCAD Command APPAUTOLOADER Sysvar

Please refer to the AutoCAD documentation for help on the commands that are supported inside of AutoCAD for this feature.

Limitations

The new loader mechanism is intended to simplify the most common App deployments. It may not be suitable for some of the more complex installation scenarios. The Autoloader component is not intended to substitute existing App loading architectures, but instead works alongside, so applications not suited to this new mechanism can still be installed using the traditional installation and registration approaches.

For AutoCAD users, CUIX files must be partial CUIX files only.

Update January 24th 2014 by Stephen Preston:

Alexander Rivilis pointed out that the formatting of the XML examples has become corrupted after Fenton copied the white paper from a Word document to the Typepad editor. (Thanks Alexander).Therefore, here is a link to the original Word doc which hopefully has less formatting issues. That said, even the Word document has been formatted for readability as a document and not for simple copying of the XML examples. Also, the formal documentation for the PackageContents.xml format is included in the AutoCAD helpfiles, and you can

download many free apps from http://apps.exchange.com and open up their accompanying XML files to see lots more examples.