This document is a guide to the process of developing an Extension server using the C++ implementation of the Altium Designer SDK, and how to install the result in Altium Designer. See An Overview of the Altium SDK document for an introduction to the SDK.

The Altium Designer software integration platform (also called the DXP software technology platform, or DXP platform for short) is based on a client-server architecture. The DXP platform allows software extension 'plug-ins' to work together and behave as a single design application, and provides you with an integrated and customizable user interface. This integrated platform provides developers a ready-built environment to develop modules that extend or enhance the existing functionality of Altium Designer.

The Altium Designer SDK provides a set of source units and source code examples that allow you to develop software extensions for Altium Designer. Altium Designer has a range of built in software extensions that provide specialized functionality such as the PCB editor and the Integrated Library server. Other extensions can be developed to enhance or expand the functionality of an Altium Designer server that has an Application Programming Interface (API), such as PCB API.

This document covers the principles and methods involved in developing a software extension server for Altium Designer using the Altium SDK, and contains three main parts:

- Introduction to Altium Designer's Client/Server architecture and software technologies
- Building and running an example project
- Installing the project extension server in Altium Designer

To extend the example project's accessibility in Altium Designer by implementing process launchers, menu items and toolbars, see Integrating an extension in Altium Designer.

This document also assumes you are familiar with using the Microsoft Visual Studio™ development tool and the .NET Framework Library, and understand C++ programming techniques such as using Classes and assemblies.
A software module developed with the SDK is generally referred to as an **Extension** or **Server** in this document, however a software extension can be implemented in a number of forms, depending on its functionality and how it integrates with Altium Designer.

Software extensions can act as design editor add-ons, design file editors, output generators, workspace extensions and more. All act as servers in the Altium Designer client-server architecture, and 'extend' its functionality – hence the moniker 'extension'. The software extension developed in the example project below acts as an add-on for the PCB Editor.

**Altium Designer's Client / Server Architecture**

Altium Designer consists of the DXP platform as the client module which hosts plug-in server applications. A plug-in server (an extension) is basically a dynamic link library (DLL) module installed in Altium Designer that has the algorithms and data structures needed to provide specific services or functionality. It can also be called a server module (or just server) within Altium Designer's client-server architecture.

The PCB and Schematic editors are examples of servers. These servers harness Altium Designer's interface elements for communication between the user and the system.

Conceptually, there is a bridge within the Altium Designer application, where the client side consists of the DXP platform module. On the other side of the bridge, the server side consists of the Workspace Manager system module, the Schematic Editor module, PCB editor, additional extension servers, and so on. The WorkSpace Manager module is a system server, while the Hole Size Editor for the PCB Editor, for example, is an add-on extension server that uses the PCB Editor's Application Programming Interface so that it can deal with the PCB objects on a PCB document.

**Services for Altium Designer**

In Altium Designer, a server module provides its specialized services in the Altium Designer environment and interfaces to the system's Graphical User Interface (GUI) through commands.

A command can be thought of as the software executing a sequence of tasks. This task can be as simple as redrawing the screen on the monitor, or it may be more complex like generating a Bill of Materials (BOM) report. Commands are the basic units of Altium Designer's GUI. A command is a packaged process launcher which is linked to a resource item in Altium Designer, for example the PCB’s **View › Area** menu item is a PCB command which has a PCB:Zoom process launcher.

A process launcher is linked to a resource item in Altium Designer and can be assigned to a key, toolbar button or a menu item. When you launch a command, a process launcher is invoked. Basically, each server has its own list of processes. These processes with their parameters are packed as process launchers that are linked to commands in Altium Designer's GUI. Whenever a command is
invoked through the GUI, the process launcher is routed to the appropriate server. This server then interprets the process launcher as a process string with parameters.

For a server to function in Altium Designer, it needs to have the APIs linked in and be accompanied by configuration files such as the server installation file (with an INS extension). The APIs provide the necessary protocols for the server to interact within the Altium Designer environment, and the server installation file provides a list of the server's available processes.

**Altium Designer SDK**

The Altium Designer SDK contains API source files, which when compiled and bundled, allow your server to function in Altium Designer. The SDK is installed as part of the Altium DXP Developer extension, and can be found in the `C:\ProgramData\Altium\Altium Designer {xxx}\Extensions\Altium Developer\SDK` folder, where `xxx` is the GUID reference for the current Altium Designer installation.

With the SDK in general, when a particular function is needed, the corresponding class must be added as an `#include` clause of your server code project. For example, to use the `MessageRouter_SendCommandToModule` procedure the SDK's EPDUtil source header (EPDUtil.h) needs to be added in the include clause. Otherwise the Visual Studio compiler will fail to build the project and report a fatal link error message in the Output window. Take a look at the files in the \\SDK\CPP\Source Code folder to see the variable and object declarations/interfaces.

**Altium Designer Object Interfaces**

An Object Interface that represents a design object in Altium Designer will have methods and properties listed (note that some interfaces will only have methods listed):

- A method is a procedure or function that is invoked from its interface. Methods of Object Interfaces are the actions an Object Interface can perform.
- A property of an object interface is like a variable; you get or set a value in a property, but some properties are read only properties, so they can only return values and cannot be set. Properties of object interfaces represent the data contained in the object that is represented by the interface.

For example the `IPCB_Component` interface from the PCB API has a Height property and two associated methods - `SetState_Height()` and `GetState_height()`. Note that the `IPCB_Component` declaration comes from the PCB API in the Altium Designer SDK and is a member of EPD.PCB.Interfaces source unit.

**Object Model Hierarchy**

In Altium Designer interface terms, a parent object interface can have child object interfaces and in turn, child object interfaces can act as parent object interfaces and have child object interfaces and so on. This forms an Object Model Hierarchy. For example, in the EDP SDK (EPD.PCB.Interfaces source), the `IWorkspace` Object Interface is the top level interface representing the Workspace manager in Altium Designer. Projects are part of the Workspace Manager, therefore an `IProject` interface is a child interface from the parent `IWorkspace` interface.

Design objects such as sheet symbols, components and buses are child objects of the document object (represented by the `IDocument` interface). This is illustrated in a simple hierarchy example below:
A relationship exists between objects in Altium Designer and the object Interfaces supported by the various APIs from the Altium Designer SDK.

- Projects and the corresponding documents are managed by the Workspace Manager - IWorkspace.
- A project open in Altium Designer is represented by the IProject object interface.
- The documents open from this project are represented by the IDocument interfaces.

The PCB documents and PCB design objects are managed by the PCB Editor and its PCB Object Model. The PCB document open is represented by its IPCB_Board interface and the design objects (for example, a pad object and a track object) are represented by IPCB_Pad and IPCB_Track interfaces.

**Altium Designer APIs**

The Altium Designer SDK has a DXP Platform API and several software product APIs:

- Client API
- Integrated Library API
- WorkSpace Manager API
- Schematic API
- PCB API

The figure below is an illustration of the relationship between objects in the Altium Designer application and the object interfaces supported by the Altium Designer SDK. This shows the PCB and Workspace Object Interfaces in Altium Designer, where the projects and the corresponding documents are managed by the Workspace Manager. An open project in Altium Designer is represented by the IProject object interface, and the documents from this project are represented by the IDocument interfaces.
The PCB and Workspace Object Interfaces in Altium Designer.

Furthermore, the PCB documents have PCB objects which are accessed by the PCB API. The open PCB document open is also represented by its `IPCB_Board` interface and design objects, for example the pad object and the track object are represented by `IPCB_Pad` and `IPCB_Track` interfaces respectively.

The PCB API provides PCB interfaces to use in server projects, and allows a programmer to modify PCB objects and their attributes from a PCB document. By way of example, you can find the `IPCB_Pad` structure in the `EDP.PCB.Interfaces` source - one of the many class dependencies loaded in most SDK example projects.

**PCBServer Functions**

When you need to work with PCB design objects in Altium Designer for example, the starting point is to invoke the `PCBServer` function (see the `EDP.PCB.Classes` source unit). With the `IPCB_ServerInterface` interface returned (`EDP.PCB.Interfaces` source), you can extract all other derived PCB interfaces that are exposed in the `IPCB_ServerInterface` interface. For example to get an access to the current PCB document, your program's code would invoke the `GetCurrentPCBBoard` method from the `IPCB_ServerInterface` interface object.

- To use PCB interfaces, you need to obtain the `IPCB_ServerInterface` object by invoking the `PCBServer` function. The `IPCB_ServerInterface` interface is the gateway to other PCB object interfaces.
- The `IPCB_Board` interface points to an existing PCB document in Altium Designer.

**SDK example project**

A good place to start in developing your own software extension for Altium Designer with the SDK is to examine and build one of the example projects provided with the SDK.

This section is a guide to building an example server that uses the PCB API to interact with an opened PCB design document in Altium Designer, where it counts the number of pad objects on the document. This extension server displays the number of pads in a dialog or in a text document in...
Note that the recommended way to start a new extension project (or edit an existing one) is by opening a new or existing Extension Publishing Document (EPD) in Altium Designer, where the DXP Developer interface acts as the central point for creating, editing, installing and publishing Altium Designer extensions.

For more information see the Altium DXP Developer documentation.

The SDK is installed along with the Altium DXP Developer extension, and the C++ SDK is located in the C:\ProgramData\Altium\Altium Designer {xxx}\Extensions\Altium Developer\SDK\CPP folder.

When the compiled extension project is installed in Altium Designer and fully configured, it will add three new menu items to the PCB menu, along with two new buttons on a new floating toolbar. To build and run the example, a copy of the Microsoft Visual Studio development tool (2010 or better) is needed.

The Visual Studio Express edition is a low cost alternative to the more elaborate Visual Studio development suites.

Setting up the project

The SDK's Addon example project is used here as a guide to the development and implementation of an Altium Designer extension. The example can be found in the \SDK\CPP\Examples\AddOn Complete folder (depending on where your copy of the SDK is installed). The folder should contain, amongst others, the files listed in the table further below.

To load the integrated structure for the Addon example:

1. Locate and open its Extension Publishing Document (Addon .EPD) in Altium Designer (File » Open and navigate to the example's VS2010 folder, with the file filter set to Extension Publisher files (*.EPD))
2. In the DXP Developer interface, install the loaded extension in Altium Designer (the Install Extension command link). This will also create a matching extension folder in Altium Designer (\Extensions\AddOn), the target for the project's DLL file.
3. Open the project code in Visual Studio (the Open Project command link in the DXP Developer interface)

For more details on editing, managing and publishing Altium Designer extensions see Altium DXP Developer.

Note that re-saving the Visual Studio project files will overwrite the source example files provided in the SDK installation itself. If you would prefer to have the installed examples remain intact as a reference, just copy the Add-on project and source files to an alternative location - for example C:\Users\Public\Documents\Altium\AD14\Extensions,
where new DXP Developer extension projects are added.

To build a server DLL that will interact with Altium Designer, Visual Studio must be able to find the Altium Designer SDK API files to compile the project. These are located in the SDK installation's \CPP\Source Code folder. If the API source path or the compiler output path needs to be defined in Visual Studio, go to the project's Property Pages dialog (Project » Properties) and change the appropriate settings.

Check that the example project is configured to write the compiled DLL to a matching folder in Altium Designer's Extensions folder (Configuration Properties | General) - in this case, C:\ProgramData\Altium\Altium Designer {GUID}\Extensions\Addon. The paths to the SDK's C++ source files can be found in the Configuration Properties | VC++ Directories section in the Properties dialog.

If necessary, use Visual Studio's project properties to include the C++ SDK source files and point the build output to the correct extensions folder.

The process of installing the extension through the DXP Developer (step 2, above) pre-configures the Visual Studio project files to match the folder structure of the Altium Designer installation. This will establish the correct source code and output file paths in advance, avoiding the need to manually configure the project properties in Visual Studio.

As with any C++ project however, the project can be opened directly in Visual Studio and inspected for reference information, without the need to open and install the extension in Altium Designer as detailed above.
Visual Studio 2012

The C++ example project files are configured for Visual Studio 2010 but can be successfully compiled in Visual Studio 2012 by loading the SDK’s alternative library file as the project dependency, if necessary.

Note that Visual Studio 2012 will offer to update the project (via a pop-up dialog) to use the Visual C++ 2012 compiler and libraries when the project is first loaded. If you selected the Don’t Update option at that time, the project can be updated later via the PROJECT » Update VC++ projects... menu command.

The SDK library file can then be changed to match the updated Visual Studio compiler and libraries. In the projects Properties Pages dialog, navigate to Configuration Properties | Linker | Input and change the Additional Dependencies entry from EDP_d.lib to EPD_2012_d.lib - the alternate 2012-compiled library file in the \CPP\Source Code folder.

![Configuration Properties](image)

For Visual Studio 2012, change the SDK library to the 2012 variant if required.

Project and source code files

An extension server as a dynamic library linking (DLL) module is constructed from Altium Designer API files (units). The essential files used for a project, as described in the tables below, are stored in the SDK C++ example project folder structure.

Add-on project files   Description
The AddOn extension project's Extension Publishing Document for the DXP Developer, which specifies the extension's publishing configuration and file paths, etc.

AddOn.vcxproj

The C++ project file.

The source and configuration files are stored in the example project's Installation and Source Code folders:

<table>
<thead>
<tr>
<th>Add-on source files</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddOn.Ins</td>
<td>The server installation/commands file.</td>
</tr>
<tr>
<td>AddOn.rcs</td>
<td>The resources file for this server project.</td>
</tr>
<tr>
<td>Commands.cpp</td>
<td>The commands unit for the server project. You also define the commands (processes) in this unit – which is the interface between the DXP platform and the extension server.</td>
</tr>
<tr>
<td>Main.cpp</td>
<td>This unit is responsible for creating and destroying the server. The command table that stores the commands is managed here.</td>
</tr>
<tr>
<td>ReportForm.cpp</td>
<td>The form unit that configures the project's report dialog.</td>
</tr>
<tr>
<td>Other files:</td>
<td>The project folder includes additional files for expanding the project – along with extra source units, these include display form and icon files.</td>
</tr>
</tbody>
</table>

Since a server Extension DLL is a library module, there is an **Exports** directive that exposes some of the functionality from the server to Altium Designer (see listing for the Add-on example project below).

In this case, a **PlugInFactory** routine from the server project, when instantiated, is exposed in the DXP platform. This routine is defined in the main.cpp unit. It takes in the **IClient** object interface which represents the DXP platform, and this same routine creates a single unique server module that stays in memory until Altium Designer is closed down.

A module-definition (.def) file is used to describe the DLL's exported **PluginFactory** function.

**AddOn.def**

```
LIBRARY  AddOn.dll
EXPORTS
    PluginFactory
```

Code snippets below refer to the Main.cpp unit for the Add-on example. This unit is where the server module is instantiated from the **PluginFactory** call, and interfacing takes place in Altium Designer (see snippet1). The main.cpp unit defines the **CServerModule** object, which is the main object representing the Extension server (snippet2). The methods and fields of the **CServerModule** class in the EDPClasses source unit are from \SDK\Source Code folder.

**Main.cpp snippets**

```
...snippet1...
```
In summary, the main.cpp unit deals with the creation of a server module/extension and sets up the server processes table (commands table). Its PluginFactory procedure passes the name of the server to Altium Designer.

**Server Processes and Server Configuration Files**

Server Processes are internal commands provided by a server which are used as the elements for process launchers that link to menus, toolbars and short cut keys in Altium Designer. So behind each menu item, toolbar button and hotkey is a process launcher, which is a wrapper around a server process, and a set of parameters and display information such as a caption string.

When you select a menu item, press a hot key or click on a toolbar button, the Altium Designer system launches the command (its packaged process launcher) by passing the server process identifier string and its parameters (if any) to the appropriate server, which then executes the action on the active document. Therefore, server processes are implemented as server process launchers which are linked to menu items, toolbar buttons and shortcut keys in Altium Designer and then are called commands. Server processes are implemented in the Commands.cpp unit of a server project.

An extension as a server module has a set of server processes, and their server process identifiers are stored in an ASCII installation file (which ends with an INS extension). Again, these server processes are wrapped as process launchers that can link to three different user interface element (resources) types. The layout of user interface elements such as the structure of menu items is defined in the resources file (which ends with a RCS extension). The module itself is a dynamic linked library (with a DLL extension) with the appropriate elements from the Altium Designer SDK.

In practice a typical server would have three files. For the Add-on example these would be, the addon.dll, the resources file addon.rcs, and the installation file addon.ins.

When an extension has been installed in Altium Designer, its compiled extension server (.DLL),
installation file (.INS) and resources file (.RCS) are implemented in a matching project folder within Altium Designer's extensions folder. An entry is also added into Altium Designer's system extension registry (ExtensionsRegistry.xml). Altium Designer will then automatically enable this new server and its processes or internal commands, so it is ready for use.

When Altium Designer starts, it reads the server installation files and stores the list of servers and their processes in an internal command table. When a command is executed (such as clicking on a tool bar button), Altium Designer checks where this button is linked to, and using the server name and its command name it then routes the command or the process launcher to the appropriate server. The server responds by executing this command if the routed command can be matched to the commands table supported by the server module.

Commands.cpp snippets

```cpp
void Command_CountPads(IServerDocumentView * const View, std::wstring * Parameters)
{
    // Check the parameters passed in
    std::wstring S;
    GetState_Parameter(*Parameters, L"DISPLAY", &S);
    // If Display = True then post results on a text document otherwise post on a dialog.
    if (UpperCase(S) == L"TRUE")
        CountPadObjects(true);
    else
        CountPadObjects(false);
}

void Command_RunAPCBProcess(IServerDocumentView * const View, std::wstring * Parameters)
{
    // Check if PCB document exists. If not stop the server.
    IPCB_ServerInterfacePtr _PCBServer(PCBServer(), false);
    if (_PCBServer == NULL)
        return;
    std::wstring param = L"";
    RunCommand(L"PCB:BoardInformation", param);
}
```

In the code sections above, the server processes (internal commands) are declared and implemented.

- All command signatures have the `IServerDocumentView, Parameters, View` and `wstring` parameters. The `View` parameter represents an opened server document in Altium Designer, and the `Parameters` parameter represents the parameters passed by the command supported by the server in the Altium Designer.
- The `InitializeCommands` procedure in the `main.cpp` unit must match the command procedure signatures in the `commands.cpp` unit (in this case, CountPads and RunAPCBProcess).
- The installation file with an INS extension must also match the commands defined in the `commands.cpp` unit, and the server module name as defined in the `main.cpp` unit must match
the **EditorName** string in the installation file that has the line **EditorName = 'AddOn'.**

See the [Server configuration files](#) document for details on creating and modifying installation configuration files.

**To Sum Up**

- A software extension is supported by its set of server processes which are implemented as commands in Altium Designer.
- The commands in Altium Designer's user interface (UI) are linked to the process launchers from the internal servers' processes tables.
- A process launcher is composed of a name, process, bitmap filename, a caption string and so on. Each process launcher is identified by a unique process identifier which includes the server name and the server process name, separated by a colon, (for example PCB:BoardInformation).
- The list of server processes is defined by the server installation file (with an INS extension).
- The layout of graphical user interface is defined in the server resources file (with a RCS extension).

**The extension in Altium Designer**

With the server project loaded in the C++ development tool, executing the Build command from the **Project»Build Addon** menu will generate a dynamic library link (DLL) file in the appropriate Altium Designer extension folder. As noted in the Project Setup section above, for the AddOn extension example this is the C:\ProgramData\Altium\Altium Designer {GUID}\Extensions\AddOn folder that was created by the initial install process in the **DXP Developer** interface.

**Server Installation File**

In general, the fundamental configuration file for enabling any extension DLL in Altium Designer is a server installation file. This should reflect the server processes declared and defined in the commands.cpp and main.cpp units of the extension project. The Command Name clauses need to include the commands defined in the other units - for example, CountPads in the case of the AddOn project, as shown below.

**Installation file: AddOn.ins**

```plaintext
ClientInsFile 1.0
Server
   EditorName            = 'AddOn'
   EditorExePath         = 'AddOn.DLL'
   EditorDescription     = 'Demonstration AddOn module'
   Version               = 'Version 6.3.0.6689'
   Date                  = '29-Dec-2012'
   HelpAboutInfo         = 'This software is protected by copyright and international treaties.'
   Copyright             = 'Copyright © Altium Limited 2013'
   Updates               = 'ADVPCB'
End
```
Register the server process

An Altium Designer Extension can be installed and registered via the *Altium DXP Developer* interface, which loads or creates a matching Extension Publishing Document (EPD). Once the Addon example project is registered via the Developer interface, Altium Designer will automatically include this new extension. Its processes (internal commands) are then ready to use in Altium Designer.

To test the Addon extension, run Altium Designer and load a PCB document that has a number of pad objects.

Executing the process in Altium Designer

To start the process manually in Altium Designer, execute the **Run Process** command from the **DXP System menu**.

In the **Run Process** dialog that opens, use the **Browse** button to select the Add-on extension from the list (AddOn:CountPads), click **OK**, then click **OK** again to run the server process. The Add-on extension checks for the presence of a PCB document, then counts the number of pads on this document. The number of pad objects found is displayed or written out to a text document.
Altium Designer GUI integration

Beyond the concept of simply calling the Add-on extension project in Altium Designer using the Run Process function, its capabilities can be extended so that it functions as an integrated feature within Altium Designer.

This is done by introducing additional server configuration files and modifying the existing files to implement suitable process launchers, menu items and toolbars. In effect, this process links the new resources into Altium Designer, or more specifically for the Add-on example project, into the PCB Editor itself.

The AddOn extension project example includes the resources file (AddOn.rcs) needed to integrate the extension's commands in the Altium Designer menus. When the extension is installed and a PCB file loaded in Altium Designer, note the two AddOn command buttons that appear in the main menu, and the addition of the Addon Menu entry under the View menu.

To read more about adding the server's process launchers in the PCB editor, adding a floating toolbar and executing a PCB command from within the server, see:

Integrating an extension in Altium Designer

Further information

For an overall look at developing and deploying extension servers using the Altium Designer SDK see An Overview of the Altium SDK.

To explore other software extension examples, see various projects in the \SDK\CPP\Examples folder of the Altium Designer SDK installation.

Source URL: https://www.altium.com/documentation/pages/viewpage.action?pageId=25073577&pageId=181824