Outpost PMM

Add-on Implementation Guide

February 2018
Version 1.1

Contents

Revision History

Date	Revision	Release	Notes
8/2/2017	1.0	V3.2	Original Doc, out for review
1/28/2018	1.1	V3.2	Supports v320c106

1 About Add-ons

1.1 Introduction

This guide will introduce you to Outpost Add-ons and describe how they work and what you need to do to develop your own Add-ons.

1.2 What is an Add-on?

Today, Outpost provides several ways for creating messages:

1. Direct enter free-form text messages
2. Copy-and-Paste or input from a text file
3. Ics213mm – A general-purpose ICS 213 message
4. NTS – National Traffic System message
5. MARS Message Maker
6. PacFORMS

The two things that the above have in common is that they:

1. solve a specific messaging problem that requires a specific message format,
2. align with how our served agencies or specific users get their work done.

An Add-on is another method with which you can create a message to be transmitted and received by Outpost, but with the message creation process outside of Outpost. Think of an Add-on as an Outpost extension. The first real add-on was PacFORMS, a series of browser-based set of forms that meet the emergency digital messaging needs of Santa Clara County OES.

With the PacFORMS functionality hard-coded into Outpost, there was interest in another way for users to develop and integrate their own messaging solutions into Outpost without a code change.

1.3 Add-ons and Outpost

Add-ons are another way to improve the message handling efficiency of communications teams who work closely with served agencies.

There are four things you need to make Add-on messaging work:

1. **Your Add-on program.** You need a tool, program, or some automated process to collect and organize your message content and get it ready for transmitting, as well as handling presenting your message on the receiving end (if required). It could be a compiled program, script, or something else to produce a message. How you do it is up to you.
2. **Add-on Config (.ini) File**. This is a file that you create to tell Outpost how you want to schedule the Add-on from Outpost.
3. **Outpost EMS (External Message Service).** Outpost EMS is delivered with each Outpost installation and includes 2 programs:
    a. **Aoclient.exe**. This program is called by your Add-on and is the first link in the chain to pass your message to Outpost. This program will then make a call to…
    b. Opdirect.exe. This program listens on the network port for and receives messages from remote message sources, such as

Aoclient.exe. Once an Add-on sends it a message, Opdirect processes and writes the message to the Outpost message database.

4. **Outpost.** Outpost is the program that interacts with the BBS for sending and retrieving packet messages. Outpost schedules your Add-on when creating a new message, sending and receiving all types of messages (Add-on messages included), and directing the message back to the Add-on to be opened in its native format.

1.4 Defining an Add-on

There are a few things that you need to do to make the entire Add-on end to end process work

1. Develop the Add-on program

Writing this program will be the bulk of your Add-on creation effort. The Add-on program needs to do the following:

- *Parse a run string to be received from Outpost*. This string will contain one or more arguments and parameters that can be passed by Outpost with specific data for the Add-on program. You define the arguments that you need, and tell Outpost how to build the Add-on run string.
- *Read a text file*. This file may be referenced by specific command line parameters that are passed to the Add-on. Your add-on may want to open a message received by Outpost, populate some form, and display it to the user in a specific format.
- *Write a text file*. When sending a message to Outpost, the Add-on must be able to create a text file that will be passed to Outpost via the Aoclient.exe program.
- *Schedule a program*. To pass a message to Outpost for sending, the Add-on must be able to schedule the Aoclient.exe program and pass it one or more command line arguments and parameters, minimally with the name of the file containing the message.

2. Create the Add-on Launch String

The Add-on can be added to the Outpost's Forms menu by including a reference to it in the Launch.local file, found in the Outpost data directory. This is usually referenced by an Include statement that points an *<add-on>.launch* file located in the Add-on directory.

3. Create the <add-on>.ini file

The *<add-on>.ini* file defines how Outpost will interact with the Add-on programs depending on the state of the message (New, Ready, Sent, Received, etc.).

A more detailed description of the entire add-on process, file formats, and command line parameters are discussed in the following sections.

1.5 Notes, assumptions, and disclaimers

The Outpost External Message Service error handling will continue to evolve over time. Most of the errors are properly trapped and reported, however, it is not 100% foolproof.

1.6 Find an Error?

If you find an error or unsure how Outpost's External Message Service is supposed to work, post a message to the Outpost Users Group or send me an email at kn6pe@arrl.net.

2 Set up Outpost to call the Add-on

This section describes how Outpost is configured to run an installed Add-on program.

2.1 Introduction

Outpost has the ability to launch a program to perform a certain task, with most of the programs related to message handling, either formatting a message or presenting a message.

A Launch Candidate is a file, program, or some other automated process that can be programmatically scheduled to perform some type of messaging task. Currently, launch candidates can be defined in one of three ways:

1. ***Legacy Launch***: This was the original implementation of Outpost. The primary method for defining these legacy launch candidates was through HTML files.

   Currently, only Santa Clara County CA RACES uses this approach with a solution called PacFORMS. These are browser-based forms and part of an application suite of forms and programs that manage the collection and presentation of messages within Santa Clara County.

   Because of the complexity of the programming involved and the choice of browsers that could be used, all of the PacFORMS solution is hard coded in Outpost, making it very difficult to change for new applications.

   NOTE: Users considering adding their own messaging solution SHOULD NOT use the HTML / Legacy Launch method.

2. ***Simple Launch:*** Outpost supports a simple format for adding executable programs as a launch candidate. These candidates can be any windows program or batch file that can be programmatically scheduled. The Outpost Ics213mm.exe program is one such example of a Simple Launch Candidate.

3. ***Add-on Launch:*** This approach provides the greatest flexibility for configuring and adding message Add-ons, and is the preferred method for defining new messaging solutions.

2.2 Launch Files

Regardless of the type, all Launch Candidates are defined in a launch file that defines what gets presented in the **Outpost > Forms** menu, and how each launch candidate gets scheduled. There are 2 types Launch files that Outpost uses:

Launch.ini	This file is delivered with Outpost. Because it will be over-written with each install process, it should not be changed. This file contains specific ICS forms that are part of the Outpost suite such as ICs213mm. It is located in the Outpost Data Directory and, on running Outpost, is read, processed, and the list of launch candidates is developed and loaded in the Forms menu.
Launch.local	This file is created by the user and has the same format as the Launch.ini file. Because it is not overwritten during the installation process, it will persist from one Outpost install to another. When adding your own launch candidates, you should add them here. The Launch.local file is also located in the Outpost data directory and, if found, is read immediately after the Launch.ini file. Its'

	contents are processed in the same manner as the Launch.ini file.

2.3 Defining a Launch Candidates

Launch files are made up of the following controls. Not all controls are allowed for all types of launch categories.

Controls	Description	Legacy	Simple	Add-on
Control Lines	One or more lines of text that guide or enhance the launch setup process.	✓	✓	✓
HTML records	Records that identify information about an HTML file to be handed off to a browser for display.	✓		
Binary executable records	Records that identify information about an existing program file to run under Windows.		✓	
Addon records	Records that identify information about an existing user–defined Add-on subsystem.			✓

Control Lines

`/ <line of text>` `# <line of text>`	Any line where the first character is a "/" or "#" character is treated as a comment line. Comment lines can occur anywhere within a launch file.
`INCLUDE <path/file_name>`	Declares the name of a file to be included. All files referenced must be in the same Launch.ini file format. Only one level of includes is permitted.
`MENU <title>`	Changes the Outpost menu name, defaults to **Forms**. Only one Menu command should be used. If more than one is listed, the last one read will be the one to be applied.
`Line`	Causes a line break on the Outpost menu

2.4 Defining Legacy Launch Candidates

HTML records are part of the original implementation in Outpost. It is limited in what it can support in terms of parameter passing and event scheduling. The format for the HTML file record is as follows:

`HTML -fn <friendly_name> -p <path\file_name> -o <parm_list>`

Where:

`HTML`	HTML record tags identify this entry as an html file.
`-fn <friendly_name>`	*Required.* The Friendly Name is loaded in the Outpost menu associated with this executable event. There should be no spaces within this string; use the "_" character instead; the load process will replace this with a space in the menu.
`-p <path\file_name>`	*Required.* The full path and file name for this

	executable.
`-o <parm_list>`	*Optional.* One or more parameter pairs to be passed with the file name. These are fixed associations and must line up to make sense, in the format: `pfvar=+opvar`. `Pfvar` PacForms Variable. One of 3 variables to be assigned. `+opvar` Outpost Variable. One of 3 variables to be passed

The list of standard PacForms variables are:

pfvar	**Description**
`ocall=+CALL`	A valid FCC call sign. Outpost makes the actual call substitution for "+CALL" at the time this PacFORMS is scheduled.
`oname=+NAME`	The name associated with the FCC call sign. Outpost makes the actual name substitution for "+NAME" at the time this PacFORMS is scheduled.
`Msgno=+MSGNO`	The message number. Outpost makes the actual Message Number substitution for "+MSGNO" at the time this PacFORMS is scheduled.

2.5 Defining Simple Launch Candidates

BIN records identify a windows program that can be set up for running by Outpost. The format for the BIN file record are as follows:

```
BIN  -fn <friendly_name>  -p <spath\file_name>  -o <parm_list>
```

Where:

`BIN`	BIN record tags identify this entry as a Windows binaryexecutable file.
`-fn <friendly_name>`	*Required.* The Friendly Name is loaded in the Outpost menu associated with this executable event. There should be no spaces within this string; use the "_" character instead; the load process will replace this with a space in the menu.
`-p <path\file_name>`	*Required.* The full path and file name for this executable.
`-w`	*Optional:* specifies that Outpost should wait for the program to complete before returning control back to Outpost.
`-o <parm_list>`	*Optional:* parameters to be passed to to the file name. These parameters are program dependent. See that programs reference manual for the command line structure.

2.6 Defining Add-on Launch Candidates

Add-on Records are very different from HTML and BIN records in that they do not specify data to be passed, but point to the configuration file that will define all run string options.

The format for the Add-on record is as follows:

`ADDON -fn <friendly_name> -a <addon_name> -t <addon_type>`

Where:

`ADDON`	non-case specific, the record tag identifying this record as an Addon (internally stored as UPPER CASE).
`-fn <friendly_name>`	*Required.* The Friendly Name is loaded in the Outpost menu associated with this executable event. There should be no spaces within this string; use the "_" character instead; the load process will replace this with a space in the menu.
`-a <addon_name>`	*Required:* Declares the Addon name; non-case specific. The ***addon_name*** is case-*insensitive* and is the specific name of the add-on. This is the same name used for the add-on configuration directory.
`-t <message_type>`	*Required:* Declares the message type, used by the Add-on program. The ***message_type*** is case-*insensitive* and defines the type of message that should be launched by the add-on. This is Add-on-specific and defined by the add-on developer.

Sample Launch.ini file

```
/ ******************************************************************
/ File:      Launch.ini
/ Desc:      Sample executable event launch file
/ Format:    Each line consists of 2 to 4 fields prefixed by a tag
/            1st field: Record Type: HTML, BIN, LINE, or MENU
/            2nd field: -fn <friendly name>
/            2nd field: -h <menu name>
/            3nd field: -p <Full path to the executable file>
/            4th field: -o <optional parameters separated by spaces>
/            Use "/" for comments or for inserting blank lines.
/ Revision:  03/29/11: Original
/ ******************************************************************
/
/Type  Friendly Name   Path to the file              Parameter list
/----  --------------  -----------------             -------------------
BIN    -fn Generic ICS-213 Message Form  -p APP_PATH\Ics213mm.exe/
LINE
```

Sample Launch.local file

```
/ ******************************************************************
/ File:      Launch.local
/ Desc:      Specific launch entries; not overwritten by a new install
/            This file is loaded immediately after Launch.ini.
/ Revision:  09/15/16: Original
/ ******************************************************************
/
/Type  Friendly Name   Path to the file              Parameter list
/----  --------------  -----------------             -------------------
HTML  -fn Messge_Form   -p C:\PacFORMS\Message.html  -o msgno=+MSGNO
BIN  -fn Notepad     -p c:\windows\notepad.exe  -o c:\Mydata\datafile.txt
/
```

```
LINE
/ include any other local addon definitions here
INCLUDE c:\Outpost_addon\aotest\aotest.launch
INCLUDE c:\Outpost_addon\alt911\alt911.launch
```

Sample addon.launch file:

```
# ********************************************************************
# File:       alt911.launch
# Desc:       Addon launch definition; forms used by the cupcert team
# Revision:   09/15/16: Original
# ********************************************************************
#
#Type  Friendly Name       addon name    addon type
#----  ------------        ----------    ---------------------
ADDON  -fn Alt911_Cactis   -a alt911     -t Cactis
ADDON  -fn Alt911_Report   -a alt911     -t Report
```

2.7 Add-on Configuration Files

Each add-on also has a configuration file that describe how Outpost will interact with the add-on, named ***<addon_name>.ini***. The file consists of a series of parameter names and a value, in the form:

`<parameter>=<value>`

There are 13 parameters that need to be setup:

Parameter Name	Description
bang-id	String, Not case-sensitive Identifies the name used by the Add-On in the message body to identify what type of message it is. It MUST be the same as the add-on name. The name also must be surrounded by exclamation points (!). Example: `bang-id=!ALT911!`
launch-created	When opening a message in Outpost, defines what Outpost should do with a locally created message (received from the Add-on) that matches the bang-id and has ***not been sent***. Assigned values: {`always|ask|never`} Example: `launch-created=ask`
launch-sent	When opening a message in Outpost, defines what Outpost should do with a locally created message that matches the bang-id and was ***previously sent***. Assigned values: {`always|ask|never`} Example: `launch-sent=never`
launch-received	When opening a message in Outpost, defines what Outpost should do with a received message (from the BBS) that matches that matches the bang-id. Assigned values: {`always|ask|never`} Example: `launch-sent=always`
msg-dir	Defines the directory where Outpost should write the message file for this Add-On to pick up. Example: `msg-dir=c:\Alt911\messages`

Parameter Name	Description
cfg-dir	Defines the directory where Outpost can find this add-on's configuration files. Example: `cfg-dir=C:\Alt911`

cmd-new	Defines what happens when calling the add-on with a message that has a message state of "New".
	This parameter is used when first selecting a form from the Outpost > Forms menu. This essentially starts the process to create a new message by the add-on.
	The value may contain literals and environmental variable substitutions. For long lines, the value string can continue on the following lines provided additional lines start with a space.
	Example:
	`cmd-new=C:\Alt911\Cactis.exe`
	`–mt {{ADDON_MSG_TYPE}}`
	`–lc {{SETUP_ID_LEGAL_CALL}}`
	`–ln "{{SETUP_ID_LEGAL_NAME}}"`
	`–mn {{MSG_NUMBER}}`

cmd-submitted	Defines what happens when calling the add-on with a message that has a message state of "Submitted". This parameter gets invoked whenever the user clicks on a message just received from the Add-on. It has not yet been transmitted by Outpost. May contain literals and environmental variables for substitutions. Multiple parameters may be included; see the `cmd-new` example. Example: `cmd-submitted=C:\Alt911\Alt911.exe`
cmd-draft	Defines what happens when calling the add-on with a message that has a message state of "DRAFT". A message is in the **DRAFT** State when a new message is opened for editing, and the user previously pressed "Save" (not "Send") on the message form. May contain literals and environmental variables for substitutions. Multiple parameters may be included; see the `cmd-new` example. Example: `cmd-draft=C:\Alt911\Alt911.exe`
cmd-ready	Defines what happens when calling the add-on with a message that has a message state of "READY". A message is in the **READY** State when a new message is opened for editing, and the user previously pressed "Send" on the message form. May contain literals and environmental variables for substitutions. Multiple parameters may be included; see the `cmd-new` example. Example: `cmd-ready=C:\Alt911\Alt911.exe`

Parameter Name	Description
cmd-sent	Defines what happens when calling the add-on with a message that has a message state of "SENT". A message is in the **SENT** State when a message was transmitted by Outpost to the BBS. May contain literals and environmental variables for substitutions. Multiple parameters may be included; see the `cmd-new` example. Example: `cmd-sent=C:\Alt911\Alt911.exe`
cmd-unread	Defines what happens when calling the add-on with a message that has a message state of "UNREAD". A message is in the **UNREAD** State when a message is retrieved from the BBS, saved in the Outpost In Tray, but before it was opened for reading. May contain literals and environmental variables for substitutions. Multiple parameters may be included; see the `cmd-new` example. Example: `cmd-unread=C:\Alt911\Alt911.exe`
cmd-read	Defines what happens when calling the add-on with a message that has a message state of "READ". A message is in the **READ** State when a message is retrieved from the BBS, and was previously opened for reading. May contain literals and environmental variables for substitutions. Multiple parameters may be included; see the `cmd-new` example. Example: `cmd-read=C:\Alt911\Alt911.exe`

2.8 Passing variables to the Add-on

When launching or sending a message to an add-on, certain information that Outpost knows but is not contained in the message can be sent to the add-on along with the message. This occurs when defining the run strings used with the cmd-<state> lines as described above.

For instance: the simplest command line is one where Outpost calls the program without any command line arguments, such as:

`cmd-draft=C:\Alt911\Alt911.exe`

In this case, when opening a message in the Draft state, Outpost will look up the `cmd-draft` parameter and schedule the named add-on using its listed run command. This may be fine but probably not real useful without passing program command line arguments. The format for passing arguments is:

`cmd-draft=<path\prgm_name> -arg1 <value1> ... –arg`$_n$ `<value`$_n$`>`

Where

`<path/prgm_name>`	name of the program to execute
`-arg`$_n$	add-on program-defined command line argument
`<value`$_n$`>`	Outpost parameter to pass to the program

Take a look at this example,

```
cmd-draft=C:\Alt911\Cactis.exe
  –mt {{ADDON_MSG_TYPE}}
  –lc {{SETUP_ID_LEGAL_CALL}}
  –ln "{{SETUP_ID_LEGAL_NAME}}"
  –sz {{ MSG_BODY_CHAR_COUNT }}
  –por
```

Here's what's happening:

1. On clicking on an Add-on message that was previous saved (as a DRAFT), Outpost looks up the `cmd-draft` parameter and plans to run the program `C:\Alt911\Cactis.exe`
2. The add-on program developer defined a couple of run-time arguments to be passed to the Add-on. These arguments are:
    - `mt` – The program needs the message type for this message (see description below)
    - `lc` – The Call Sign currently selected in Outpost
    - `ln` – The name of the user
    - `sz` – The message sized as stored in Outpost
3. All arguments are passed with a dash "–"followed immediately by the argument.

NOTE: Argument names are defined by the developer. They can be anyhing you want.

4. In this example, all of these run time arguments require a parameter (or Outpost environmental variable) that Outpost will substitute with the real Outpost value when building the add-on run command.
    a. All parameters for these run time arguments must be bracketed with the double curly opening {{ and closing }} brackets.
    b. The program developer could also define run time arguments without parameters.

The following are the parameters available for substitution by Outpost. Variable substitution is dependent on when the Add-on is scheduled. Scheduling occurs by two methods:

1. **Outpost > Forms, Menu Click**. When you first click on an Add-on from the Forms menu, you are essentially initiating the creation of a new message. Variables used here can only be used with the **`cmd-new`** parameter, and have a check mark (✓) in the Menu-Click column below.
2. **Outpost > opening a listed message**. If an add-on message is listed in the message list, opening it by double-clicking it or using the **Open** button will cause Outpost to check on the message status, and then check if the message should be opened with any of the other **`cmd-<state>`** parameters. These parameters have a check mark (✓) in the Msg Open column below.

		Availability	
Environmental Variable	**Value**	**Menu-Click**	**Message Open**
The following variables are global in nature, and change only when they are updated in Outpost.			
SETUP_ID_LEGAL_CALL	Setup > Identification > Legal > User Call Sign	✓	✓
SETUP_ID_LEGAL_NAME	Setup > Identification > Legal > User Name	✓	✓
SETUP_ID_LEGAL_PFX	Setup > Identification > Legal > Message ID Prefix	✓	✓
SETUP_ID_TAC_CALL	Setup > Identification > Tactical > Tactical Call Sign	✓	✓
SETUP_ID_TAC_NAME	Setup > Identification > Tactical > Additional ID Text	✓	✓
SETUP_ID_TAC_PFX	Setup > Identification > Tactical > Message ID Prefix	✓	✓
SETUP_ID_ACTIVE_CALL	Setup > Identification > Legal/ Tactical Call Sign (whichever is selected)	✓	✓
SETUP_ID_ACTIVE_NAME	Setup > Identification > Legal/ Tactical Additional ID Text (whichever is selected)	✓	✓
SETUP_ID_ACTIVE_PFX	Setup > Identification > Legal/ Tactical Message ID Prefix (whichever is selected)	✓	✓
SETUP_BBS_CONNECT_NAME	Setup > BBS > Connect Name	✓	✓
SETUP_NEW_MSG_NUMBER	Setup > Message Settings > Msg Tab	✓	
The following variables are from the Message itself			
MSG_BBS	BBS field in the message		✓
MSG_FROM_HEADER	Full address (not comments) in "From:" field. Ex: If From: is user@host.domain, then value is "user@host.domain" Ex: If From: is "Big User" <user@host.domain>, then value is "user@host.domain"		✓
MSG_FROM_LOCAL	Part of From: address before the "@" Ex: If From: is user@host.domain, then value is "user"		✓
MSG_FROM_FQDN	Part of From: address after the "@" Ex. If From: is user@host.domain, then value is "host.domain"		✓

		Availability	
Environmental Variable	**Value**	**Menu-Click**	**Message Open**
MSG_FROM_HOST	Part of From: address after the "@" and before the first "."; null if From address is local only. Ex. If From: is user@host.domain, then value is "host"		✓
MSG_FROM_DOMAIN	Part of From: address after the first "." After the "@"; null if from address does not include a domain Ex. If From is user@host.domain, then value is "domain"		✓
MSG_TO_HEADER	All address(es) in the "To:" field; will be full addresses if any are in a user@host.domain format.		✓
MSG_SUBJECT	Subject: from the message		✓
MSG_DATETIME_HEADER	Date/Time from the "Date:" from the received message; null if the message was created locally		✓
MSG_DATETIME_OP_SENT	Date/Time the message was sent by Outpost to the BBS; null if the message has not been sent (it was received, or it was created but not sent).		✓
MSG_DATETIME_OP_RCVD	Date/Time the message was received by Outpost from the BBS; null if the message was not received (it is locally created)		✓
MSG_NUMBER	The message number (for outgoing messages)		✓
MSG_LOCAL_ID	The local message ID assigned (if any) (for incoming msgs)		✓
MSG_BODY_CHAR_COUNT	Character count of message body		✓
MSG_STATE	Text of: new, ready, sent, unread, read, draft, abandoned, deleted		✓
MSG_ORIGIN	Text of: created, received		✓
MSG_FILENAME	Name of the message file being passed (no path info)		✓
The following variables are set in the Add-on definition file and applies to all messages associated with a specific add-on.			
ADDON_MSG_TYPE	Type of message (set by Launch.ini, launch.local or included launch files)	✓	

		Availability	
Environmental Variable	**Value**	**Menu-Click**	**Message Open**
ADDON_MSG_DIR	Absolute path to the directory where Outpost will place files to be read by the Add-On (defined in Add-On config file <add-on-name>.ini). Null if message does not contain a !bang-id!	✓	✓
ADDON_CFG_DIR	Absolute path to the directory where the configuration files for the add-on can be found. These files include: <add-on-name>.ini and <add-on-name>.launch.	✓	✓

2.9 Add-on Considerations for Receiving Parameter

1. All variables are passed to an Add-on on the command line.
2. Variable substitution is independent of the add-on syntax. In other words, each add-on may have its own command line syntax to accomplish the same thing. Those different syntaxes are independent of Outpost and can easily be configured in the add-on configuration.
3. If a command line variable may contain spaces, such as the legal name or the filename, then the command line is formatted as:

```
add-on.exe -lc {{SETUP_ID_LEGAL_CALL}}
  -ln "{{SETUP_ID_LEGAL_NAME}}"
  -fn "{{MSG_FILENAME}}"
```

   Double Quotes should be used to quote any substitutions which might result in spaces being included. In fact, it is safest to quote everything that is not certain to be a numerical value (such as the character count).
4. Variable substitution rules are driven based on where the Add-on is called.
   a. ***Menu_Click***. Only SETUP_ and ADDON_ environmental variables are allowed.
      1. MSG_ variables are will not be substituted since no message exists at this point.
      2. NOTE: A "SETUP_NEW_MSG_NUMBER" variable exists that allows a message to be created to be assigned an Outpost Message Number.
   b. ***Message Open***. All SETUP_, MSG_, and ADDON_ environmental variables are allowed with the following two exceptions (exclusive to Menu_Click events):
      1. SETUP_NEW_MSG_NUMBER
      2. ADDON_MSG_TYPE

2.10 Add-on Program considerations

!! What your ADD-ON needs to do:

1. The options you define in the add-on.ini file imply that your Add-on can consume and needs them. Develop the parsing mechanism to pick up and store command line options to satisfy your program's requirements.

2. It is recommended that you store your application in a directory other than the standard Windows programs directories to avoid any conflicts with Windows' User Access Controls. For instance, you may name your directory:
   `C:\MyPrograms\Outpost_addons\<MyAddonName>`
3. Store your <add-on>.ini file in the directory where your Add-on resides.
4. In the Launch.Local file, add an **Include** statement to reference to your <addon_name>.ini.

3 Set up the Add-on to call Outpost

This section describes how the Add-on needs to be designed to pass a created message back to Outpost.

3.1 Introduction

Building your Add-on is totally up to you in terms of development language and tools. However, at some point, you will want to pass your message to Outpost for action.

The Outpost/Add-on approach specifies a simplified interface for your Add-on to pass a message to Outpost. Once your Add-on is built, there are four things you need to do:

1. Copy the program **Aoclient.exe** into your program directory. This program is installed with Outpost. If you are distributing your Add-on to a wider audience, the author of Outpost grants permission to include this program with your installer.
2. Configure Outpost to have Opdirect.exe running.
3. When your message in the Add-on is ready, the Add-on must create a text file that contains the message body.
4. The Add-on runs Aoclient.exe with a run string that you create to pass all the necessary parameters (along with the message file name) that will define your message in Outpost.

3.2 Aoclient program setup

This program is the first step in getting your message loaded into Outpost. Aoclient.exe has 2 modes of operation.

1. ***Interactive Mode:*** When running the program without any command line options, the program opens in Interactive mode.
2. ***Command Line Mode:*** When one or more command line options are passed to Aoclient.exe, the program opens in Command Line mode. All menus and controls are disabled and unavailable to the user. On completing the message hand-off process, the program terminates.

Here is the sequence of events for command line mode:

From your Add-on:

1. Create your message and any message parameters that your program supports.
2. Create a text file of your message.
3. Build the Aoclient.exe run string,
4. Schedule Aoclient.exe with your command line arguments and parameters.

At Aoclient.exe:

1. Looks for and detects the command line options.
2. Confirms that the `-f` option (file name) is minimally set.
3. Extracts the run time parameters, opens and reads the message file, and formats a message with all included command line options.
4. Aoclient.exe makes a network connection call to Opdirect.exe, and hands off the message.

5. When Aoclient is done, it creates a PASS / FAIL semaphore file for the Add-on to read to determine if the submittal was a success. The Add-on then exits.
6. Meanwhile, Opdirect writes the message into the Outpost database.

Setup

You will need to do the following only once to set up Aoclient.exe:

1. Copy the Aoclient.exe program from the Outpost Programs Directory into your Add-on program directory.
2. Once Aoclient.exe is in your Add-on's directory, double-click on it to run it.
3. Select ***File > Preferences***, then the ***Add-on ID*** tab. Enter the name of the Add-on.
4. Select **File > Preferences**, then the **Network** tab. If your Add-on is running on a different PC than Outpost, but on the same LAN, enter the IP address of the Outpost PC where Opdirect.exe is running. If it is running on the same PC as Outpost, do not change any of the defaults.
5. Press OK when Done. Aoclient is now set up.

3.3 Outpost Setup

The Opdirect.exe program listens for network connections from programs like PacFORMS, Ics213mm.exe, and Aoclient.exe who want to pass their messages to Outpost. Opdirect.exe must be running for your Add-on process to function correctly.

Setup

1. From Outpost, select **Tools > Message Settings**, **Adv** tab.
2. Check the box "Automatically start the Opdirect External Message Service"
3. Press OK, and restart Outpost. You will see Opdirect.exe running in the System Tray.

3.4 Create the message text file

The add-on's message is passed to Aoclient.exe as a standard ASCII text file. How you create the file is up to you and the capabilities of your development environment.

!! What your ADD-ON needs to do:

1. Create the message in your add-on in whatever manner you have defined.
2. When ready to send your message, format your message in standard 7- or 8-bit ASCII. These files are typically marked as <some_name>.txt, but you can name it whatever you want. All lines should be terminated with either a CR or CRLF.

3.5 Build the Aoclient run string

All other message parameters are passed as Aoclient.exe command line parameters. One of the parameters is the file name containing the message text.

!! What your ADD-ON needs to do: Command Line Options

All parameters are passed to Aoclient.exe as command-line options. These options are:

Cmd Line Option	Description
-a <addon_name>	REQUIRED. Name of this Add-on.

Cmd Line Option	Description
	Example: -**a ARKSTAT**
-f <file_name>	REQUIRED. Name of the file containing the message created by the Add-on to be passed to Outpost. *Example:* -**f c:\Outpost_addon\Arkstat\ArkstatMsg.txt**
-b <bbs_connect_name>	OPTIONAL. Name of the BBS to be set for this message. *Default:* currently selected BBS *Example:* **-b W6XSC-1**
-t <destination_list>	OPTIONAL. List of one or more destination addresses to be set for this message. If this option is not set, then Outpost will leave this field on the message blank, or use the Default Destination (if set). *Default:* **<none set>** *Example:* **-t "KN6PE, kd6tcmv@arrl.net"**
-s <subject>	OPTIONAL. Subject to be set for this message. If Message ID is set, Outpost will automatically insert the Message ID ahead of the subject. If this option is not set, then the field will be blank. To ensure a new message ID is created (if this is an original message creation event), then precede the Subject with "++" (no quotes); this will force Outpost to load the next message ID onto the subject line. *Default:* **<none set>** *Example:* **-s "++DeAnza ARK Status, OP Period #2"**
-u	URGENT. OPTIONAL. Sets the message status to URGENT. If this option is not set, then Outpost will leave the message as not Urgent. There is no parameter that follows this option. *Default:* **Not Urgent** if not added to the command line *Example:* **-u**
-y	Message Type. OPTIONAL. Sets the message Type. The string is one of these three values: [PVT | NTS | BUL]. *Default:* PRIVATE (PVT) *Example:* **-y NTS**
-p	NOPOP. OPTIONAL. Determines if message will be opened automatically once it arrives in Outpost. If set, Outpost will not open the message. *Default:* Not set. Outpost will automatically open the message *Example:* **-p** **(**Outpost will not open the message)
-dlt	DoLinkTest. OPTIONAL Causes Aoclient to initiate a test connection with Opdirect. As a result of this test, either an OpdPASS or OpdFAIL file will be created in the directory where Aoclient resides.

!! What your ADD-ON needs to do: Run a program

1. Build the Aoclient.exe command line. For example, the following code example will set up and schedule Aoclient.exe, for instance:

   VB

   ```
   Program = "Aoclient.exe"
   RunLine = " –f RunCmd.Text"
   Result = Shell(Program & " " & RunLine)
   ```

   C++

   ```
   nRet= ShellExecute(0,"open",Prgm,Runline,0,SW_SHOWNORMAL)
   ```

2. Check if the call to Aoclient.exe was successful. The best way to do this is by (i) checking that Aoclient completed, and then (ii) test for the presence of the Aoclient files "OpdPASS" or "OpdFAIL", where:

   OpdPASS – The message was successfully passed to Opdirect.
   OpdFAIL – The message was not passed to Opdirect.

   A fail indication may be caused by:
   1. Aoclient not pointing to the correct IP address for Opdirect
   2. Opdirect is not running.

   NOTE: The above are samples for demonstration purposes only. Refer to your programming language documentation for the exact method for scheduling a program.

3. Watch for Aoclient.exe error messages in the event that program has a problem passing your message to Outpost.

4 The Add-on Process

The following describes the Add-on end to end process.

4.1 Outpost, Program Start / Initialization

The following occurs each time Outpost is run. If changes to any of the Launch or Add-on files are made, exit out of Outpost, and then run Outpost again to pick up all changes.

Create Launch Menu (the Outpost Forms Menu)
- Initialize Launch Data
  - Read the Launch.ini
  - If found, read the Launch.local
  - If found, process all INCLUDE statements; add their content as well
  - Load the Launch Data Array (7 fields per launch candidate)
- Initialize Add-on Data
  - Find all Add-ons in the Launch Data Array
    - If found, read the <addon_name>.ini file
  - Load contents into the Add-on Data Array (13 fields per Add-on)
- Populate the Outpost > Forms Menu from the Launch Array

End.

4.2 Outpost > Forms, Click

Selecting an entry from the Forms Menu will create a NEW message (implies there is no message body). The Add-on run string is based on the Menu item being an HTML, BIN, or ADD-ON message to be created.

If HTML,
- Retrieve the correct Launch Array entry
- Build the run string (html path and option string)
- Set up the Msg ID
- Make the run string substitutions
- Find the Default Browser
- Run the browser with the run string

If BIN,
- Retrieve the correct Launch Array entry
- Build the run string (program path and option string)
- Run the program with the run string

If ADD-ON,
- Retrieve the correct Launch Array entry
- Retrieve the correct Add-on Data Array entry
- Get the Open Action (based on the loaded <addon.ini> Config File)
- If Open Action is “ALWAYS” or “ASK” (with a YES answer),
  - Build the Run String (based on Message State)
  - Make the run string substitutions
  - Run the Add-on with run string

4.3 Outpost > Open Message Form

The message form will open either because it was “auto-popped” open by an incoming Add-on message (via Aoclient and Opdirect), or the Packet Operator manually opened the message from the Outpost message list.

If a PacFORM,
- Process PacFORM Message
- If Open Action is “ALWAYS” or “ASK” (with a YES answer),
  - Create the message file (embed parameters in the file)
  - Build the run string (program, message file name)
  - Run the browser with the run string

If ADD-ON,
- Get !AddonBangID!. Look up the Addon in the Launch and Add-on Data Arrays.
- Get the Open Action (based on the loaded <addon.ini> Config File).
- If Open Action is “ALWAYS” or “ASK” (with a YES answer),
  - Get Run String (based on Message State)
  - Make run string substitutions
  - Create the message file
  - Schedule the Add-on