Creating Custom OutpostX Forms

Contents

1 Introduction

OutpostX Forms provide a way to create structured messages without requiring a separate program or custom code for each form. A form definition describes the information the operator enters, how that information is represented in an OutpostX message and, optionally, how the received information is displayed on a standard printable form.

The important distinction is that the form itself is not transmitted. OutpostX converts the information entered on the form into a normal text-based OutpostX message. That message can then be sent using the same packet-radio methods used for other OutpostX messages.

At the receiving station, OutpostX recognizes the form information embedded in the message. If the receiving station has the corresponding form definition, then the message can be reconstructed and displayed using the original data-entry form. If a PDF template is also provided, OutpostX can place the received values onto the printable form.

This gives an OutpostX Form three useful representations:

- Data-entry form — optimized for entering the information.
- Packet message — a text representation suitable for transmission.
- Printable form — an optional PDF representation of the received information.

These three representations do not have to look alike. The data-entry screen should be designed for efficient operator entry; the packet message representation is compact and readable; and the PDF representation can reproduce the appearance of an official or locally developed paper form.

1.1 Forms are still OutpostX messages

The OutpostX Forms system was designed so that structured forms do not create a separate message transport. When the operator selects Create Message (forms data entry page), the form data becomes part of an ordinary OutpostX message.

For example, a portion of an ICS 213 message might contain:

```
!OPXFORM:FEMA.ICS213:1!
1-INCIDENT NAME: Flooding Risk
2-TO: Logistics
3-FROM: Operations
4-SUBJECT: Portable Generator Request
...
!/OPXFORM!
```

A receiving copy of OutpostX that recognizes FEMA.ICS213 can reconstruct the form. A station without the form definition can still read the information as ordinary text.

1.2 What kinds of forms can be created?

The forms capability is not limited to ICS forms. An organization might create forms for:

- local situation/status reports;
- resource requests;
- damage assessments;
- alternate 9-1-1 reports;
- shelter or facility status;
- event or public-service communications; or
- other structured information routinely exchanged over packet.

A custom form does not require a PDF. For many locally developed forms, the OutpostX data-entry form and readable packet representation may be all that is needed. A PDF is useful when the information received must be displayed or printed using an existing standardized form if required by the receiving served agency.

1.3 Components of a Form

The two files define how OutpostX Message uses forms:

Component	Purpose
MYORG.MyForm.opxform	**Required.** Defines the form identity, data-entry fields, packet representation, and optional PDF mappings with a user-friendly markup language.
MYORG.MyForm.pdf	**Optional.** Provides the graphical/printable representation onto which received values can be placed. This could be an official agency form (FEMA) or a custom form you have created yourselves.

The base name consists of the Form Organization followed by its name. For example:

- .opxform – Create this file
  - FEMA.ICS213.opxform FEMA, ICS 213 Message form
  - CUP.ALT911.opxform Cupertino, local 911 message form
  - YRORG.SITSTAT.opxform <YRORG>, SITSTAT form
- .PDF
  - FEMA.ICS213.pdf FEMA, Renamed to ensure the base file names match
  - CUP.ALT911.pdf Cupertino, custom form that Cupertino ARES created
  - YRORG.SITSTAT.pdf <YRORG>, SITSTAT form

NOTE: Use the same base filename for both the .opdform and .pdf files.

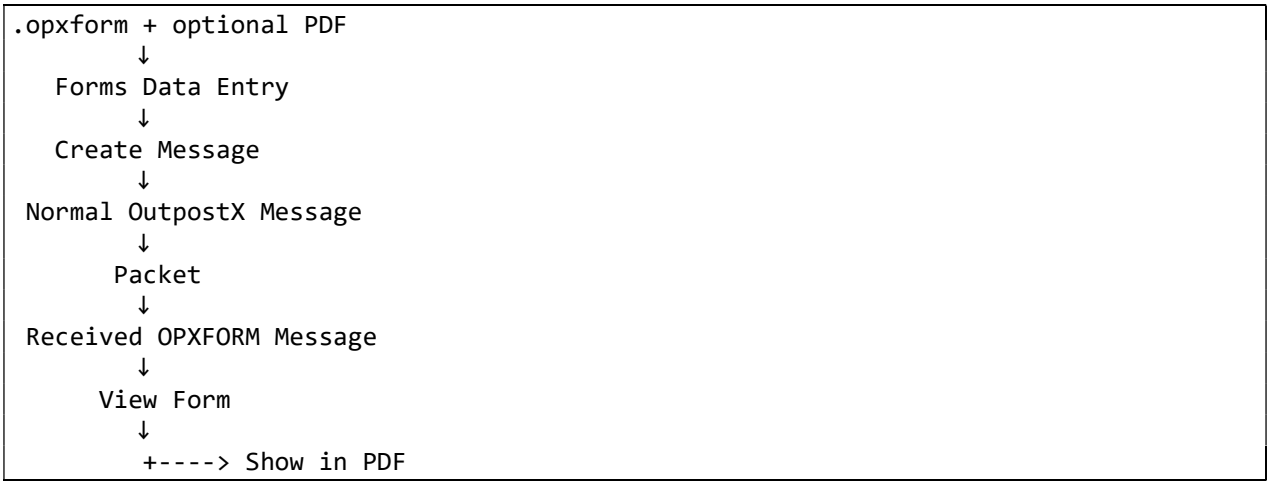
NOTE: Typical 'fill-in-the-blanks' forms work fine. A lot of the features were added as we experimented with different forms. If your form has a special need, please let me know and we can check to see if it makes sense to add it.

IMPORTANT: The .opxform form defines the data, packet representation, and optional PDF presentation mapping. **No programming should be required to add a normal custom form.**

1.4 Message Form Lifecycle

The lifecycle is intentionally simple. The sending station uses the .opxform definition file to collect information and create the message. The message then travels through OutpostX exactly like other packet messages. At the receiving station, the same form definition allows OutpostX to reconstruct the structured information. The optional PDF provides a presentation layer for viewing or printing the received information.

If the goal is to pass messages on structured forms between sites, then both ends need the definition for form reconstruction and .pdf file, while the underlying message remains readable without it.



2 Building a Form Solution

2.1 Start with the operational form

Identify the actual information that must be entered. For ICS 213, it is:

1 Incident Name (optional)
2 To
3 From
4 Subject
5 Date
6 Time
7 Message
8A Approved By
8B Signed
8C Position

Important rule: Define the information, not every line and box on the paper form.

2.2 Define the input fields

- Create one definition for each logical value.
- Start with the simplest field and build outward, or copy from a form that is similar.

IMPORTANT: get the data-entry form working before touching the PDF.

2.3 Define the wire format

Decide whether the raw packet message needs to be human-readable.

For ICS 213:

7-MESSAGE: Request two portable generators.

For a compact OutpostX-only form:

7: Request two portable generators.

Rule: If non-Outpost users may receive the message, then use human-readable labels.

2.4 Test Create Message

Open: Menu → Forms → your form

Enter data and press **Create Message**.

Inspect the resulting body.

Make sure you see the expected:

```
!OPXFORM:MYORG.MyForm:1!
...
!/OPXFORM!
```

Do this before adding PDF support. This will isolate form-definition problems from PDF-layout problems.

2.5 Add the PDF

Place the matching PDF in the Forms directory and add the PDF definition/mappings.

IMPORTANT: A PDF is optional. Don't use or create one unless the form needs a graphical/printable representation.

2.6 Calibrate the PDF

Generate the ¼-inch coordinate grid.

Then:

```
Grid
  ↓
Estimate x/y
  ↓
Edit .opxform
  ↓
Reload Definition
  ↓
Show in PDF
  ↓
Adjust
```

2.7 Test end-to-end reconstruction

Send the message through packet.

At the receiving end, Open Message → Actions → View Form → **Show in PDF**

Verify that the received form contains the same information as the originating form.

3 Using .opxform Files

3.1 Input Section

The input section defines the data-entry form presented to the operator. It contains the form title and a list of fields. Each field represents one logical piece of information that will be collected and, normally, included in the resulting OutpostX message.

For example, this definition creates a required multiline field for the message text:

```
"input": {
  "title": "ICS 213 General Message",
  "fields": [
    {
      "id": "message",
      "wire_id": "7",
      "wire_label": "MESSAGE",
      "type": "multiline",
      "hint": "Enter message",
      "required": true,
      "show_label": true,
      "rows": 10
    }
  ]
}
```

Input Field Elements

Element	Description
id	Unique internal name for the field. This name is also used to associate the value with PDF mappings and other references within the form definition. Example: "message".
wire_id	Short, unique identifier for the field in the transmitted form data. It should remain stable once a form is placed into use. Example: "7".
wire_label	Human-readable packet label; what actually is displayed on the input form for this field. Example: "MESSAGE".
type	Identifies the type of data-entry control OutpostX presents to the operator. ***See Field Types*** below.
rows	For a multiline field, specify the appropriate number of input rows to display.
required	true or false. When true, the operator must provide a value before the form can be used to create a message.
hint	Data entry guidance (hint). A text string that is presented until the field is filled in, example: "Enter message".
show_label	true or false. Controls whether a label is displayed on the data-entry form.
default	Supplies an initial value for the field. The operator may overwrite the value. Special values such as $CURRENT_DATE and $CURRENT_TIME may be used.

Field Types

OutpostX currently supports these input field types:

Type	Use
text	A single line of text. Suitable for names, locations, subjects, identifiers, and similar short values.
multiline	Multiple lines of text. Suitable for message bodies, comments, descriptions, addresses, and other longer entries.
checkbox	A true/false selection. Suitable for Yes/No or selected/not-selected information.
date	A date entry field.
time	A time entry field.

Design guideline: If your form has other controls not listed here, then send me an email and copy of your form

Design guideline: Create one input field for each logical value that needs to be collected. Do not create fields merely to reproduce every line, box, or visual element appearing on a paper form.

3.2 Transport Section

The transport section controls how the form's field values are represented in the text of the OutpostX message.

```
"transport": {
  "show_labels": true,
  "omit_empty": true
}
```

Element	Description
show_labels	true or false. When set to true, human-readable wire_label values are included with the corresponding field values in the packet message. See below.
omit_empty	true or false. When enabled, fields containing no data are omitted from the packet representation. This can reduce message size, particularly for forms containing many optional fields.

Human-Readable Transport

With **show_labels** set to **true**, a field may appear in the message as:

```
7-MESSAGE: Request two portable generators.
```

This representation is useful when the message may need to be read by an operator or station that does not have the corresponding OutpostX form definition.

A more compact representation can be used when human-readable labels are not required:

```
7: Request two portable generators.
```

3.3 PDF Section

The optional pdf section defines how form values are placed onto an existing PDF document. The PDF is a presentation layer; it does not define the data that is transmitted.

A form does not require a PDF. Use this section only when the received information needs to be displayed or printed using a standardized or locally developed graphical form.

For example:

```
"pdf": {
  "template": "CUP.ICS213.pdf",
  "fields": [
    {
      "id": "incident_name",
      "page": 1,
      "x": 180,
      "y": 58,
      "width": 390,
      "height": 18,
      "font_size": 10
    }
  ]
}
```

PDF Elements

Element	Description
template	Filename of the PDF template associated with the form.
id	Identifies the form field whose value is to be placed at this location on the PDF.
page	PDF page on which the value is placed. Page numbering begins with page 1.
x, y	Field position: referenced from the top left. This is form specific.
width, height	Defines the available area into which the value is rendered.
font_size	Font size used to render the value.
multiline	When enabled, allows the value to occupy multiple lines within the defined area. See an example in the next section (FEMA.ICS213.opxform).
wrap	When enabled, wraps text within the available field width.

Advanced PDF Presentation

PDF mappings may also control what value is displayed and under what conditions.

Element	Description
source_id	Uses the value from another form field rather than the field identified by the mapping's id. This allows the same entered value to appear in more than one place on the PDF.
show_if	Displays the mapped value only when the specified condition is true.
prefix	Adds fixed text before the displayed value.
suffix	Adds fixed text after the displayed value

For example, the ICS 213 definition uses the approved_by value to produce a signature notation only when the signed checkbox has been selected:

```
{
  "id": "signed",
  "source_id": "approved_by",
  "show_if": "signed",
  "prefix": "/s/ ",
  "page": 1,
  "x": 306,
  "y": 430,
  "width": 220,
  "height": 16,
  "font_size": 8
}
```

Design guideline: Keep data collection, packet transport, and PDF presentation separate. Define a value once in the input form and reuse it when necessary for PDF presentation rather than creating duplicate fields.

3.4 ICS 213 Message Form, .opxform file example

```
{
  "opxform_version": 1,
  "form_id": "FEMA.ICS213",
  "form_version": 1,
  "name": "ICS 213 General Message",

  "input": {
    "title": "ICS 213 General Message",
    "subject": {
      "source": "field",
      "value": "subject"
    },
    "fields": [
      { "id": "incident_name", "wire_id": "1", "wire_label": "Incident Name", "type": "text",                                        "required": false },
      { "id": "to",            "wire_id": "2", "wire_label": "TO",            "type": "text",      "hint": "enter a name, position", "required": true  },
      { "id": "from",          "wire_id": "3", "wire_label": "FROM",          "type": "text",      "hint": "enter a name, position", "required": true  },
      { "id": "subject",       "wire_id": "4", "wire_label": "Subject",       "type": "text",                                       "required": true  },
      { "id": "date",          "wire_id": "5", "wire_label": "DATE",          "type": "date",      "default": "$CURRENT_DATE",      "required": true  },
      { "id": "time",          "wire_id": "6", "wire_label": "Time",          "type": "time",      "default": "$CURRENT_TIME",      "required": true  },
      { "id": "message",       "wire_id": "7", "wire_label": "Message",       "type": "multiline", "rows": 10,                    "required": true  },
      { "id": "approved_by",   "wire_id": "8A","wire_label": "APPROVED BY",   "type": "text",                                       "required": true  },
      { "id": "signed",        "wire_id": "8B","wire_label": "SIGNED",        "type": "checkbox",                                   "required": false },
      { "id": "position",      "wire_id": "8C","wire_label": "POSITION",      "type": "text",                                       "required": true  }
    ]
  },
  "transport": {
    "show_labels": true,
    "omit_empty": true
  },
  "pdf": {
    "template": "FEMA.ICS213.pdf",
    "fields": [
      { "id": "incident_name", "page": 1, "x": 180, "y": 58,  "width": 390, "height": 18,  "font_size": 10 },
      { "id": "to",            "page": 1, "x": 180, "y": 75,  "width": 390, "height": 18,  "font_size": 10 },
      { "id": "from",          "page": 1, "x": 180, "y": 105, "width": 390, "height": 18,  "font_size": 10 },
      { "id": "subject",       "page": 1, "x": 54,  "y": 146, "width": 230, "height": 16,  "font_size": 10 },
      { "id": "date",          "page": 1, "x": 450, "y": 146, "width": 100, "height": 16,  "font_size": 10 },
      { "id": "time",          "page": 1, "x": 522, "y": 146, "width": 80,  "height": 16,  "font_size": 10 },
      { "id": "message",       "page": 1, "x": 54,  "y": 180, "width": 486, "height": 205, "font_size": 12, "multiline": true, "wrap": true },
      { "id": "approved_by",   "page": 1, "x": 162, "y": 430, "width": 220, "height": 16,  "font_size": 10 },
      { "id": "signed",        "page": 1, "x": 306, "y": 430, "width": 220, "height": 16,  "font_size": 8,
        "source_id": "approved_by", "show_if": "signed", "prefix": "/s/ " },
      { "id": "position",      "page": 1, "x": 486, "y": 430, "width": 220, "height": 16,  "font_size": 10 }
    ]
  }
}
```

4 Where the Files Go

The place all forms in the OutpostPMX data directory, runtime Forms subdirectory. Place all pairs together:

```
forms/
    MYORG.MyForm.opxform
    MYORG.MyForm.pdf
```

NOTE: the following forms are supplied by OutpostPMX and are updated each and every time the program runs:

FEMA.ICS213. OutpostPMX-distributed form

IMPORTANT: Do not modify an OutpostPMX-supplied form and retain its official filename. A future installation may restore the distributed version. Copy it to a new form identity when creating a local variation.

5 Test Your Form

Short acceptance checklist:

1. Form loads and appears under New → Forms.
2. Input fields, defaults, and required fields work.
3. Create Message produces the expected OPXFORM text.
4. Packet Subject and form Subject remain correct.
5. Show in PDF looks correct, if applicable.
6. Send the message over an actual packet path.
7. Receiving OutpostX recognizes View Form.
8. Received values match the sent values.
9. Received PDF looks correct.
10. Temporarily remove the .opxform and verify the message is still readable as ordinary text.

6 Common Mistakes

Problem	Better Approach
1. Creating one field for every printed line	Create one field for each logical value
2. Making input form look exactly like PDF	Optimize input for data entry
3. Long arbitrary wire IDs	Use short, stable IDs
4. Duplicating a value for another PDF location	Reuse the existing value
5. Putting /s/ into transmitted signature name	Add it as PDF presentation
6. Changing official FEMA.* files	Copy to your organization's form ID
7. Starting PDF calibration before form works	Prove Create Message first
8. Testing only locally	Perform an actual send/receive test
9. Assuming PDF is mandatory	Use one only when needed

7 Where to Go Next

You now have enough information to create and test a basic OutpostX form. More complex forms may require system defaults, multiline transport handling, conditional PDF presentation, field reuse, signature notation, compatibility/version management, or detailed graceful-degradation testing.

Need more detail? See the **OutpostX Forms Authoring Guide** (***PENDING***) for the complete definition and behavior of each element.

Need to…	**Authoring Guide**
Understand every .opxform element	Sections 4–7
Design human-readable packet data	Sections 7–8
Add PDF output	Section 9
Calibrate PDF fields	Section 10
Add conditional PDF presentation	Section 11
Fully test a form	Section 12
See the complete ICS 213 example	Section 13
Review design practices	Section 15