REUSABLE SOLUTIONS

BASED ON

MVC MODEL FORMS

Version 1.0
August 22nd, 2006

Revision and Related Document Change Log

Document Revisions

Revision Number	Date	Page(s) Affected	Comments	Author
1.0	August 24th, 2006	All	Initial Version	Miroslav Lukic

Related Documents

Revision Number	Date	Document Title	Comment
1.4	May 5, 2006	High Level Design for UUG – Web Quote Form	

Adobe

Table of Contents

1 Introduction

1.1 Document purpose

The purpose of this document is to describe specific reusable solution components that have being implemented within Adobe Consulting around "MVC model" data entry forms.

1.2 Used case scenario / requirements

Typical used case scenario solved by this solution type is:

- Form is (mostly) used for data entry purposes and not printing
- The end user requires data entry form to be used in both **online and offline mode**.
- In most cases (mainly due to large amount of data that needs to be entered), majority of the data entry process is done **offline** with "online" data submission and retrieval
- Data entry form is **highly complex**:
    - It contains multiple pages (usually 10 or more)
    - It contains multiple repeatable sections (dynamic subforms) spread across multiple pages
    - Due to offline form usage, data validation and other relatively simple business rules are implemented within the form. In most cases, data needs to be validated before it is submitted to server side components
- Data entry form needs to offer reasonably well client side performance.

1.3 Solution types

In terms of complexity, there are couple types of solutions that are built using this concept:

- Standalone XDP based PDF form (with or without Adobe LiveCycle Reader Extension Server)
- "Run time assembled" XDP based PDF form (created using Adobe LiveCycle Forms with/without Adobe LiveCycle Reader Extension Server)

The next couple sections will provide more details regarding these solutions.

Adobe

2 Standalone XDP based PDF form

2.1 Solution components

In typical deployment, standalone form is developed using Adobe Form Designer 7.x and then saved as "Dynamic PDF Form File". Depending on the end user requirement, "dynamic PDF" is then "reader enabled" using Adobe LiveCycle Reader Extension Server and final "reader enabled" PDF is usually deployed on the web server. The end user accessed PDF form through web browser and either:

- Fills out and submits form online
- Saves forms offline and submits it in online mode (from either web browser or standalone Reader)

Note: In this type of architecture, online submission is handled through either web services or Java servlets. However, there is an opportunity to add Adobe LiveCycle Forms to process data and/or merge it with other forms.

The main solution components are:

- Standalone XDP based PDF form
- Adobe LiveCycle Reader Extension Server version 7
- Adobe LiveCycle Designer 7.x

2.2 Standalone XDP based PDF form

Due to complexity described in section 1.2, standalone PDF form is typically implemented using "MVC model" applied to XFA form. The main advantages of this approach are:

- The form offers much better performance comparing to the traditional approach of displaying form with all pages being visible and presentable at all time. The main performance problem with typical WYSIWYG approach is time required to redraw the form when dynamic sections (subforms) are added/removed. Current version of Designer/Reader (7.x) redraws the whole PDF regardless of where dynamic subforms are added/removed

  Note: It is expected that overall performance will be improved using different "redraw" algorithm with Reader 8.x.

- Panels (pages) are only instantiated at run-time (when required) so the end user can see **only one "panel" at time.** It allows a complete control of what the end user can see in the document (as one would expect with a MVC model). Since only one panel is presented at time, client side performance is much better (Reader needs to "deal" with only subset of the "complete" form)
- Each "panel" can include dynamic subforms that can span over multiple physical pages
- All panel data is persisted when moving between "panels" and/or saving the document (note that additional code need to be written to perform this functionality)

The main reusable components in the form itself is script object named (in most cases) **controller.** It is used to implement majority of functionality required to support this type of solution such as:

- Ability to move between different panels while showing only one panel at time

- Ability to save/load data between panels
- Ability to retain/retrieve data when Reader is closed/opened in offline mode
- Ability to perform data validations

2.2.1 Controller script object

Following table contains some of the major functions implemented in the typical **controller** object:

Function name	**Description**
init()	Called from **form:iniitalize** event and its purpose is to initialize complete form.
executeAction()	Used to call function named forward() to move between different panels
forward()	Used to move between panels (called from executeAction())
savepage()	Save data for the current page into in-memory data store array or "global variable (depending whether binding is used or not)
loadpage()	Load data for the page from the in-memory data store or "global" variable
getPageNum()	Get a page number from its name
scanPages()	Get the list of all body pages from instance manager
retrieveDataStore()	Retrieve content of the dataStore array from data_Store page
saveDataStore()	Save the content of the dataStore array to dataStore page
updateQuickViewList()	Update the drop-down list (pop-up menu) with available selections
updateAllRelevantPanels()	Update array of selected panels based on the end user selection
showAllPages()	Show all pages in the form – typically used for print preview purposes
hideAllPages()	Hides all pages in the form with the exception of the currently selected page
validatePanel()	Used to perform panel level validation
removePanels()	Remove particular panels
saveDataStoreComplete()	Save the content of the dataStore array
Debug function	Various functions used to write to Acrobat console

Following table contains some of the major variables implemented in the typical **controller** object:

Variable name	**Description**
dataStore	Array to contain data content for each panel

bBindingUsed	Used to identify whether data binding is used
strBindingData	Used to hold complete XML data when binding is used
arrPanelDisplayNames	Array of all panel display names to be shown in the quick navigation list
arrPanelPageNames	Array of names that correspond to "page no"

Detail description of each function and its source code can be found in Adobe Consulting Perforce repository (**Halifax.can.adobe.com:1800**) in multiple projects such as:

- "High level design for UUG Web Quote Form" (section 3) contains detail description of each function used in controller object as well as some other script objects. Note that this document contains some UUG specific implementation details that may not be used in other projects **(\Professional_Services\Project_Delivery\U\UUG\UUG001\Documentation\ High Level Design.doc)**

- Implemented XDP/PDF forms for multiple project such as:

    - UUG form(s) - found in **\Professional_Services\Project_Delivery\U\UUG\UUG001\Development\UUGForms**

    - AEGIS form – found in **\Professional_Services\Project_Delivery\A\AEGIS\N.AES002 - AEGIS Insurance Application Form\Development\Data-Entry Wizard**

    - Fidelity form(s) – found in **\Professional_Services\Project_Delivery\F\Fidelity\FRIP**

Note that depending on the client requirements there are slight differences between controller script object implementations. However, the core functions are (more or less) used in each implementation.

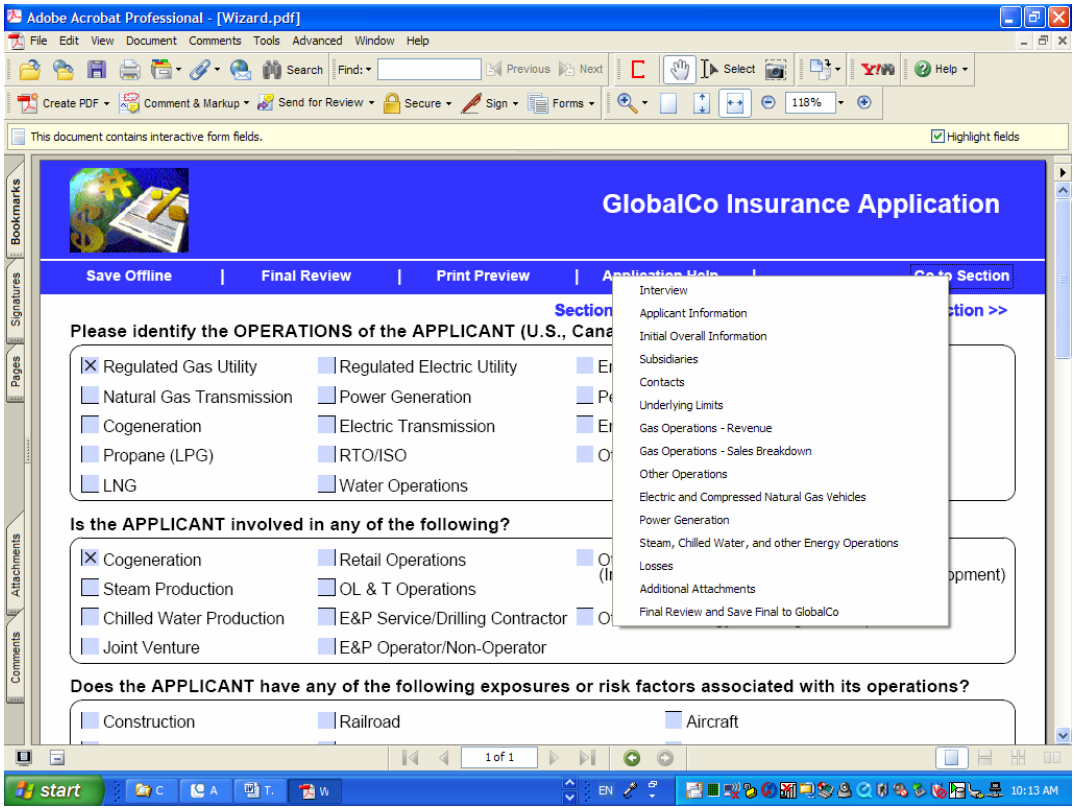
Note: For additional details regarding controller object, please consult files mentioned above

2.2.2 UI interface

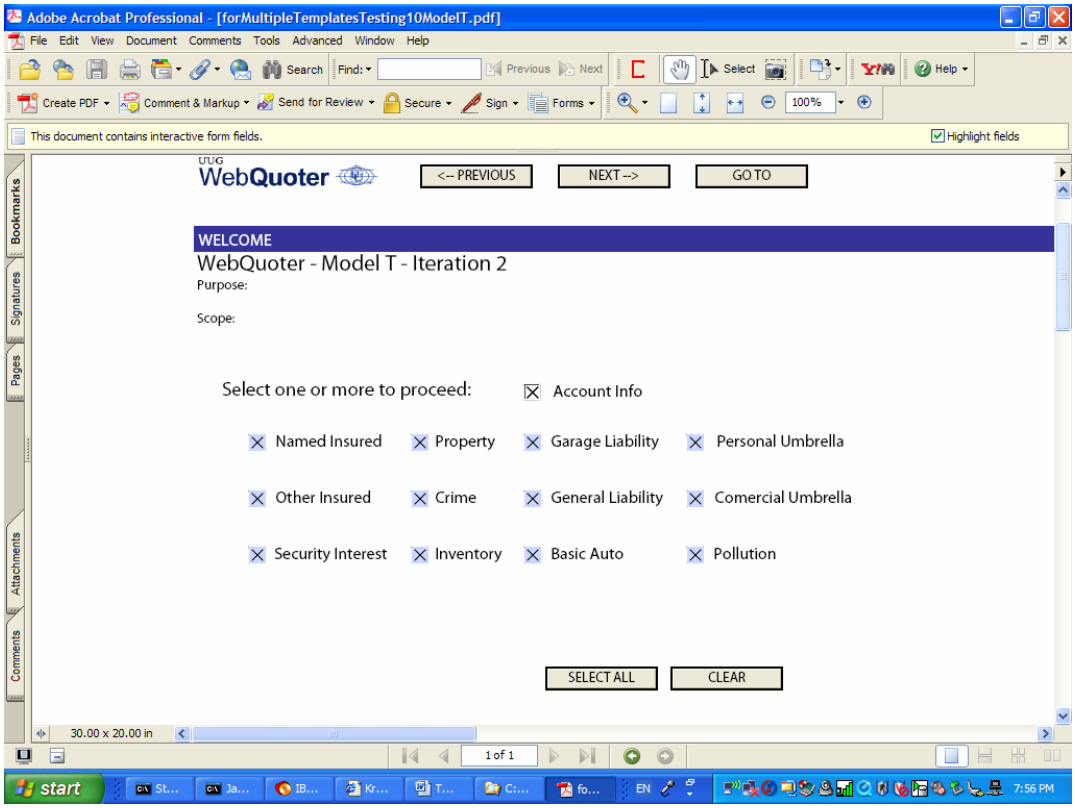
Typical UI interface consist of couple logical panels:

- Welcome page (interview) panel is used to decide what other panels should be displayed to the end user. Typically this panel consists of multiple check boxes/radio button and underlying Java script logic used to determine, based on the end user selection, what additional panels need to be displayed. The logic is implemented through Java script in the form itself due to its offline usage. Here are couple typical panels implemented for various customers:

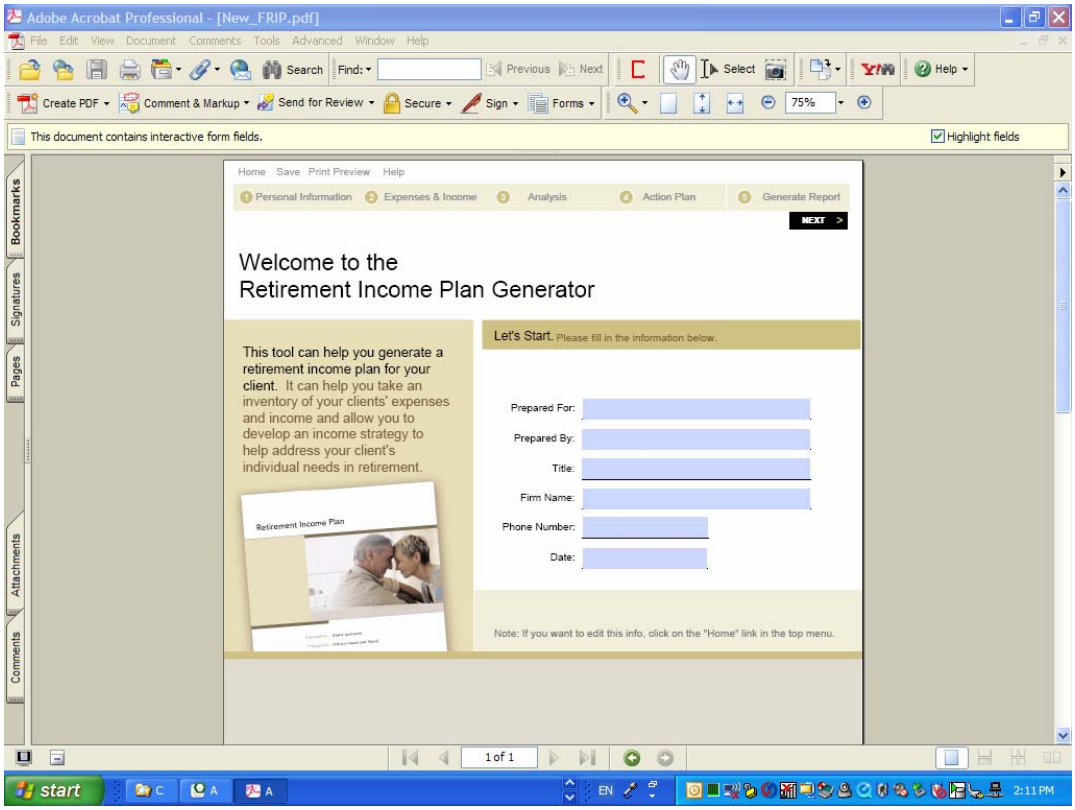
    - AEGIS – the end user is asked to select one or more combinations of Operation/Coverage and based on his/hers selection various panels are available. Panel navigation is achieved either using "Next" and "Previous" button and/or "Go To" button that provides short-cuts to various panels. In addition, this panel is typically used to provide some other application specific functionality such as ability to print, submit data and so on



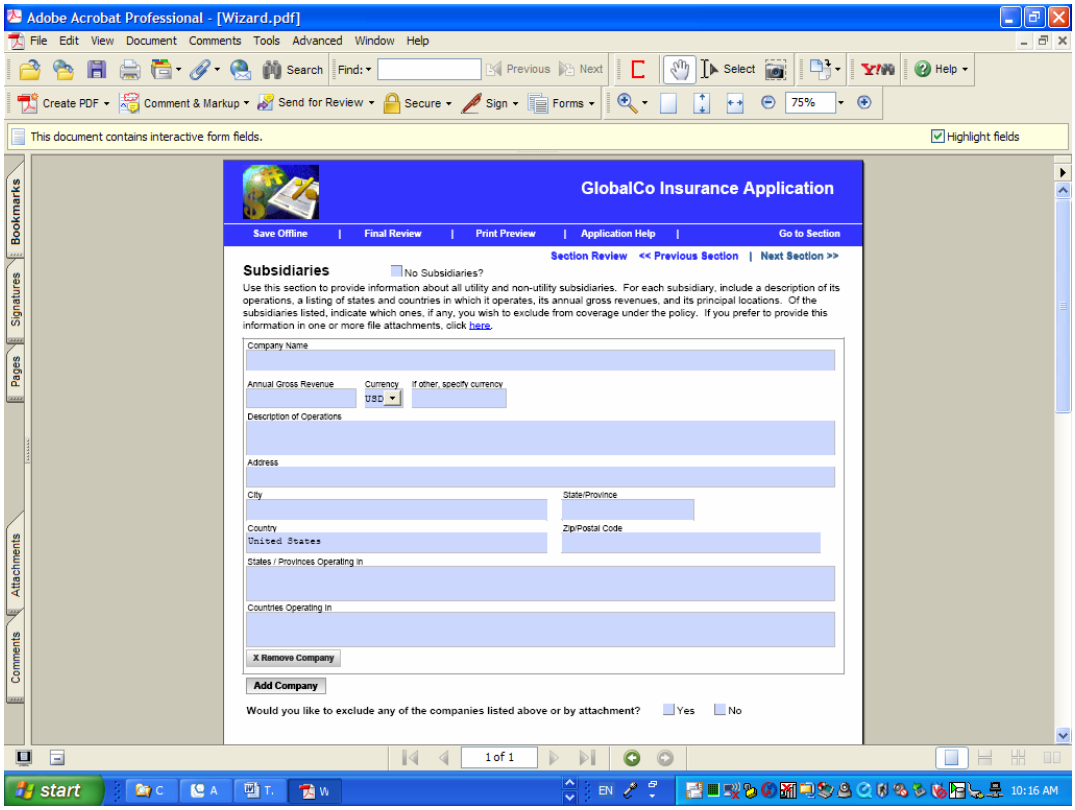
- UUG – the end user is asked to select one or more coverage types for particular client. Underlying business logic is much simpler since there is "one to one" relationship between check boxes and panel. "Select All" and "Clear" button allow the end user to quickly select/deselect all check boxes on the first panel. Note that UUG form is still under development and it is expected that additional functionality (such as an ability to save offline, submit/retrieve data will be placed on the panel)



- Fidelity – the end user can switch panels using button on the top of the initial panel (similar to tab control). In this particular instance, the end user can access any panel at any time. In addition to "panel specific buttons", there are some other buttons (such as "Save", "Print Preview") to provide generic functionality.



- Data entry panels – as far as "MVC mode data entry form" is concerned these panels appear as independent entities. Since each panel is shown on its own (instead of all of them being shown at the same time), performance is much better. Each panel can span to multiple physical pages and it can contain dynamic subforms (such as the one shown below). Note that in typical implementation, "panel header" containing navigation buttons, header and title is retained between panels.



- "Hidden panels" (such as page_datastore) are used to perform application specific functionality (such as saving data in offline mode and so on).

2.3 Adobe LiveCycle Reader Extension Server

It is used to provide standard product functionality – "reader enable" final PDF before it is finally deployed. Since PDF is enabled at design time, this can be accomplished through either standard HTML interface and/or through custom code (typically Java classes integrated in the overall application).

2.4 Adobe product used

- Adobe LiveCycle Designer 7.x
- Adobe LiveCycle Reader Extension Server 7.x (optional)
- Adobe Reader 7.x

Note that following Adobe products could be used to complement overall solution:

- *Adobe Policy Server – its main usage would be to "expire" document after certain date (according to the policy) to prevent end users from using outdated documents*
- *Adobe LiveCycle Forms – its main usage would be to accept submitted data and merge it with different templates*

2.5 Challenges

The main challenges are related to development of very complex "standalone PDF form". Some of them are documented in the next couple paragraphs.

2.5.1 Form performance

Forms designed using "MVC model" offer much better performance comparing to the traditional approach of designing form with all pages being visible and presentable at all time. However, there is still some performance problems related to:

- Current version of Adobe Designer/Reader has memory leaks when using dynamic forms. Its main cause is use of **instancemanager** ability to show/hide dynamic subforms and need for Reader to redraw whole template once new subform is added/deleted. It is expected that this functionality will be rectified with Reader 8.x. In the meantime, there is no real solution to this problem except starting/stopping Reader to reclaim memory. Some of the "work around" approaches used to "hide" this particular problem are:
    - Design form with more smaller, simpler panels as opposed to single more complex panel
    - Implement function to periodically ask the end user to save data to prevent possible data loss
- Current version of Adobe Reader saves PDF in increments resulting in relatively big PDF files. Newer version of Reader (such as 7.05 and later) partially rectify this problem so the end user should be encouraged to upgrade to the most recent version of Adobe LiveCycle Reader Extension/Acrobat Reader. It is expected that this problem will be fixed in the later releases. When submitting to server side components, it is recommended to submit only XML data and not PDF itself.

2.5.2 Form printing

Forms based on "MVC model" are mainly designed for data entry purpose. In case of print preview requirement (such as typical "fill and print" solutions), there are couple options:

- Print preview form client side (using functions showAllPages() and hideAllPages() in controller script object). There are couple things to be aware of:
    - This approach requires all panels to be previewed at the same time and it can result in large form being displayed in Acrobat Reader. Depending on the number of pages and end user workstation configuration (mainly available memory), it could take long time to render the form. Additional code (such as making all editable objects on the form read only) would need to be written to prevent the end user from using this form for data entry purposes
    - Functions mentioned above implements page breaks between different panels
    - Additional functionality may be written to provide "printer friendly" version of the data entry form (such as replacing dropdown with fields, formatting objects and so on).
- Print preview form server side by submitting data to Adobe LiveCycle Forms and merging with different template used for "print preview":
    - This approach requires Adobe LiveCycle Forms to be part of the overall architecture and it may be hard to justify unless other usage for Adobe LiveCycle Forms could be found (such as ability to produce different type of forms)

- It enables clear separation of "print preview" and data entry form. "Print preview" form typically does not require any validation except (in some cases) some calculations.
- It requires the end user to be online in order to print form

- Print preview form can be accomplished by manually exporting data from "MVC model" form and importing into another PDF ("print preview form") residing on the end user workstation

2.5.3 Code complexity and maintenance

The code to support "MVC model" is relatively complex and it typically involves steep learning curve for the new form developer. Some of the main challenges are:

- Understanding functionality of controller object
- Additional code needs to be written to perform data validation. In typical case data validation can be performed at couple levels:
  - Field level
  - Panel level
  - Document level (before final submit)

  In most cases, document and panel level validation routines are used with very minimal field level validation.
- Since "MVC model" presents one panel at the time any "inter panel rules" typically require values to be passed as either "global" fields (typically set in script object) or form variables. These values often need to be maintained while Acrobat Reader is opened and stored/retrieved before/after Reader is opened
- Forms based on "MVC model" save/restore data differently depending on whether data binding is used. Additional if-else logic needs to be implemented (such as the case with UUG forms) since (in most cases) schema definition file (XSD) is not ready before form development had started. It is often the case that form designer start with complete unbound form and then binds it later
- Depending on the complexity, some of the panels are forms itself with its own set of business rules. For some customers (typically in the insurance industry) individual panels are typically "state specific" and may contain different objects/rules depending on the state
- Due to its heavy offline usage (in some cases), additional business logic is typically implemented in the form itself in order to validate as much as possible.

2.5.4 Single master page

Current "MVC model" is restricted to usage of single master page. Therefore, common master page should be designed for all panels. This restriction had lead to the implementation of "custom" page sizes in certain cases to accommodate for the most "complicated" panels.

Note: It would be probably possible to implement multiple master pages through code.

2.5.5 Presentation and business logic

Due to its offline usage, majority of the business rules are implemented as various Java script functions/objects throughout the form. There is no clear separation of presentation and business logic mainly due to client requirements. This is typically driven by business requirements to be able to catch as many errors as possible and avoid sending "invalid data" to server side components.

Adobe

3 "Run time assembled" PDF

3.1 Solution components

The main difference between "run time assembled" and "standalone" PDF is how and when final PDF is assembled. Typically, "run time assembled" PDF are used in the following cases:

- Customer would be prefer to maintain individual panels as separate templates due to its complexity (for example some of the panels are "state specific" (typical requirement in insurance industry) and contain "state specific" object/business rules)
- Final PDF form may require only portion of available panels (forms). For example, only panel for particular state should be included in the final PDF.
- The panels are complex with its own set of independent business rules so they can be treated as independent XDP templates.

The main solution components are:

- Standalone XDP based PDF form
- Assembly ("form stitching") component
- Adobe LiveCycle Forms 7.x
- Adobe LiveCycle Reader Extension Server 7.x
- Adobe LiveCycle Designer 7.x
- Front end component (such as portal)

Typical process for assembling final PDF templates involves following steps (note that some of the steps such as invoking Adobe LiveCycle Reader Extension Server are optional):

- Based on the end user selection made in the portal or some other front end application (such as choosing primary state) appropriate XDP templates are retrieved
- **Assembly component** is invoked to assemble multiple XDP templates (retrieved in the previous step) together. This operation typically involves combining multiple XDP files into single "master" XDP file.
- XML data (to be merged with "master" XDP) is retrieved from back end system(s)
- **Adobe LiveCycle Forms component** is invoked to render "master" XDP including XML data file retrieved in the previous step and create "interactive" PDF file ("PDFForm")
- **Adobe LiveCycle Reader Extension Server component** is invoked to apply usage rights to PDF document created by Adobe LiveCycle Forms
- Final **PDF document** is sent back to the calling component (typically downloaded to local hard drive). It is expected that the end users will typically used this as data entry point for multiple clients.

*Note: For detail description regarding this type of implementation please refer to "High level design for UUG Web Quote Form" found in Adobe Consulting Perforce repository (***Halifax.can.adobe.com:1800***) at:*
\Professional_Services\Project_Delivery\U\UUG\UUG001\Documentation\High Level Design.doc

In most cases, Adobe server side component are developed and implemented as Java packages (series of Java classes packaged into JAR file) that can be imported into other server side component, referenced and used. This approach ensures that any potential changes to Adobe server side API's in the future will not affect the components built by customer (such as "portal"), rather changes will be made directly to Adobe class components that are part of the package.

3.2 "Run time assembled" PDF form

As far as the end user is concerned, there is no visual difference between "run time assembled" and standalone PDF form (described in paragraph 2.2). Once final PDF is assembled, it contains same components and outlook as the one described in section 2.2. Therefore, the same reusable objects (such as controller script object) are used.

Note: In the example forms mentioned in section 2.2.2), UUG form is "run time assembled" while AEGIS and Fidelity form are developed as standalone.

3.3 Assembly components

This component is developed by Adobe Consulting to assemble multiple XDP templates into single "master" template that can be then passed to LiveCycle Forms. It accomplishes this by performing various XML manipulations on the XDP templates. XDP templates are treated as XML document that are loaded and manipulated using XML parsers. Note that depending on the user requirements, different methods have being implemented in Assembly components for different projects. Typically, these methods are used to:

- Adobe Assemble multiple XDP templates into single "master" XDP
- Replace subforms within XDP itself

The main reasons for "custom components" are current product limitations:

- Adobe LiveCycle Forms allows single XML data stream to be merged **only with single XDP template**
- Adobe LiveCycle Assembler can not assemble together multiple XDP based PDF files

3.4 Adobe LiveCycle Forms

This component performs standard product functionality of merging XDP template with XML data stream and rendering it as "PDFForm" to the browser session that has requested it.

3.5 Adobe LiveCycle Reader Extension Server

This component performs standard product functionality to reader enable PDF file. Since PDF is typically enabled at run time, this is usually accomplished through Java classes integrated in the overall application.

3.6 Front end component (portal)

This is custom component used to perform one of more steps such as:

- Determine what XDP templates need to be assembled into "master" XDP
- Retrieve XDP templates from repository (usually file system or document management)
- Invoke various other components (such as Adobe LiveCycle products and/or other custom components)

This component is typically written and managed by customer.

3.7 Adobe product used

- Adobe LiveCycle Designer 7.x
- Adobe LiveCycle Reader Extension Server 7.x
- Adobe LiveCycle Forms
- Adobe Reader 7.x
- Custom components

Note that following Adobe products could be used to complement overall solution:

- *Adobe Policy Server – its main usage would be to "expire" document after certain date (according to the policy) to prevent end users from using outdated documents*

3.8 Challenges

As far as challenges with this type of solution, all of the ones related to front end data entry form (described in 2.5) apply in this case as well. The other set challenges are related to the usage of server side ("back end") components.

3.8.1 Server side response time

Final data entry form is created from individual components so server side components need to be executed every time new PDF is requested. This adds significant amount of time to initial response time (especially if PDF needs to be "reader enabled" as well). Couple possible ways to improve performance are:

- Create often used template combinations as PDF files and store them in repository (such as document management system). Additional logic is then added to front end component (portal) to pull "pre made" template from repository instead of invoking LiveCycle component at run time
- Implement all functions in Assembly component using SAX parser as opposed to XML DOM (at the present time some of the functionality has already being written using SAX)

3.8.2 "Custom" assembly component

Assembly component (described in 3.3) is one of the key components of the overall solution. This component is maintained by Adobe Consulting and requires period updates such as:

- Additional development and QA time to ensure that component is compatible with newer version of Adobe LiveCycle Forms

- Potential rework when/if customer has decided to upgrade LiveCycle Forms
- Rework required when/if certain functionality becomes available in standard product set

It is anticipated that newer version of Adobe product set would be offering some of the functionality that is currently performed by Assemble components such as:

- Adobe LiveCycle Designer 8 will be capable of handling "form fragments" as separate XDP templates. That would allow form designer to maintain separate XDP's and merge them as design/run time.
- Adobe LiveCycle Assembler 8.x should be capable of assembling multiple XDP based PDF templates. This would allow us to stitch PDF templates using standard product instead of stitching XDP templates using custom components. Note that performance test should be run to verify response time in this particular case.
- Newer version of Adobe LiveCycle Forms may provide an ability to merge single data stream with multiple XDP templates (similar to Adobe Output Server functionality)

Depending on the product releases, functionality provided by Assembly component (or parts of it) may be replaced by standard product set. As a result of it, potential upgrades to the existing customer using Assembly component will require changes in the server side components.

Assembly component provides its functionality by parsing XDP as XML documents and performing various "transformations" on it (such as inserting subforms at appropriate level). Therefore, it imposes another set of restrictions (depending on the method invoked) mostly around naming convention. For example, when assembling multiple XDP file together, top level subform in each XDP must be uniquely named across all XDP files to be assembled in order for "master" XDP to function properly. At design time, form designer must be aware of these restrictions in order to be able to create functional "master" XDP. Some of the recommendations are:

- Maintain all script objects and form variables into single XDP (usually the first one to be assembled)
- Implement methods that do not assemble script objects/form variables from any other templates but the first one

3.8.3 Server side submission

One of the common requirements is an ability to submit data to some server side component for further processing. Note that once data is submitted from browser session, PDF context is lost from browser window so it would need to be recreated by calling server side component (as described above). Possible "work around" is to, instead of submitting to Adobe LiveCycle Forms, submit to web service component that would be used to process data and then return data back to the calling component.

3.8.4 Coding and testing

In order to test overall form functionality, "master" XDP has to be created and opened in Form Designer. In addition, all "inter panel dependent rules" can be only implemented in the "master" XDP and not the individual templates. In most cases, "test harness" component used to execute assembly component and create "master" XDP is provided to the customer.

Once "inter panel dependent rules" are implemented, they are typically moved into one of the panels.

3.8.5 "Duplicate ID"

One of errors that have happened sporadically when opening "master" XDP (assembled from multiple XDP files) in Form Designer is:

Failure, duplicate id 'xxxxxx' found in the template model. Ids must be unique.

This error is probably caused by Form Designer encountering "non unique" ids across multiple XDP files. The only work around so far is to redo XDP that has caused the error. Note that this error is currently being investigated by Adobe Consulting / Enterprise Support.

4 Comparison

The purpose of this section is to describe difference between two approaches described in previous sections.

"Run time assembled" PDF approach add layer of complexity and challenges (described in section 3.8) due to usage of server side components. However, some of the advantages of this approach are:

- Enables customer to maintain panels as separate templates. This approach is best used when there are different rules/implementations for the same panel – for example "state specific" coverage panels in insurance industry. Depending on the complexity, it may be advantageous to implement each panel as separate XDP instead of adding "state specific" logic in the same panel

- Depending on the business requirements, only relevant panels can be assembled into final PDF resulting into smaller PDF being sent to the end user

- Adobe LiveCycle Forms provides additional functionality such as:

    - Data entry form can be pre-populated by merging XML with XDP/PDF template before final form is server to the end user (note that same functionality can be implemented for "Standalone PDF" by invoking web service before form is finally presented to the end user)

    - Allows data to be easily shared with other form(s). Typically, data collected in this type of forms is used to create another documents (some of them legal) that are presented to the end customer