Appian Tempo and SAIL

Serafeim Papastefanos
spapas@gmail.com

Features of Appian Versions

6.1	Data Stores	Query Rules	Write to Data Store	CDT Import/Export
6.2	CDT Designer	Multiple Analytics Engines		
6.5	Tempo	Appian Mobile		
6.6.x	Start Form Expression	Paging Grid	Increment constant	
6.7.x	Looping functions			
7	Some enchancements from Tempo			
7.2	SAIL	Constructing Data Type Values	Records	Tempo Reports
7.3	Enchancements to the above			

New concepts

- Tempo Interface
  - News
  - Tasks
    - Good old user tasks
    - Probably cannot be filtered (ask Appian)
    - Use Tempo Forms
  - Records
  - Reports
  - Actions
    - Process Launchers
    - Grouped by application

Record

- The data of a process
- https://forum.appian.com/suite/wiki/73/Record_Design
- How do records simplify traditional process modeling? We expect many data-centric use cases based on records to simplify the design and performance, and testing of process models. *As an example, many formerly "long-running processes" can be shortened by maintaining data in a record instead of in the process instance.*
- Are records a substitute for process dashboards? *Yes. Records are a new way of conveying process information, as well as other business data, to users using Appian's new SAIL framework.* T
- Record source can be either
  - Entity (This is the modern way)
    - Backed up by a traditional RDBMS
  - Process (Closer to traditional Appian concept)
  - Data from a service (more advanced)
    - For instance a web service

Record components

- Facets
  - Add filter to records
  - Show only processes started today
  - Show Applications for a customer
- Record List View
  - Only 100 can be displayed
    - Use Tempo Reports to do actual reporting
  - Add facets to do filter and find the one you want
  - Name - description - time
- Record DashBoards
  - Sail Summary DashBoard is the main page of the record
  - You can add other dashboards to view other data for each record
  - Could replace the Process Information page or just link to it
- Related Actions
  - Quick tasks for process records
  - Start process models by passing parameters for entity and service records
  - Modify XYZ Application data
    - Only available to XYZ administrators

News

- Processes create Events to news feeds
  - Post Event to Feed - For instance a user completed a task or started a Process
    - Feed, Message, User
  - Post System Event to Feed - For instance a process has been Escalated
    - Feed, Message, Source (text)
  - Post Hazard to Feed - Highlights an Event
  - Create comment to Event
- Can have different news feeds per process
  - Visible to specific groups
- Could be used instead of emailing
- Link to Record
  - “A new XYZ Application has been submitted”
  - XYZ Application entity is persisted
  - Link to the dashboard of that Entity based record
- Users can reply to events
- Users cannot post to the news feed unless they have specific rights

Tempo Reports

- Present data to user
- https://forum.appian.com/suite/wiki/73/Tempo_Report_Design
- How do Tempo reports differ from Portal reports? Tempo reports can easily be modified through their expression, be viewed on mobile devices and the broad set of web browsers supported by Tempo, and report on data from tasks, records, and relational databases. Portal reports can only be viewed from the Portal and can only report on data from process models, processes, and tasks.
- Can be saved as Task Reports to be viewable from Tasks tab of tempo
- Sail Dashboard
  - Define the data with load() / with()
  - Represent it with SAIL
- Old portal reports *can* be easily converted to Tempo reports !
-

Tempo Forms

- An appian form can be saved as a Tempo form
  - Tempo forms can be opened from a variety of browsers (not just IE) and from mobile apps
- Javascript *cannot* be used on Tempo forms
  - No custom javascripts
  - No ajax
  - This is actually a good thing
- Other components that cannot be used in Tempo forms
  - Tabs, *Grids*, Reports, Messages, Custom CSS, Confirmations
  - Use paging grid instead of grid
  - Use paging grid instead of cascading dropdowns
- Everything could be converted to be a Tempo form
  - Some things might be different for the user
  - A universal interface is very important - no more customizations per app

Examples

1. See the Appian forum for example usage

2. See the User Group Management Application

How should we model our processes

- A single Record / Entity that keeps all the data of the process
- No more long running processes
  - A process shouldn't last for more than a few days
  - Add Quick Tasks to cancel processes / add escalations to cancel tasks
  - The data of the process will be updated after each small appian process finished - and will be refreshed from the RDBMS when a new process instance is started
- What about special cases ?
  - FirstData: Just end your process when the email to the FirstData leaves and start a new one when you get a reply
- No more Process Dashboard / Info => Record Dashboard
  - Select the record you want and "act" on it - pass its protocol number to the child process
- We are already doing this: The "status" and data of a Tempme application is kept in the database - many Appian processes read from and write to it for a *single* application
  - Now the tools exist to do it better!!!
- Always use Tempo forms
  - Complexity should go to a series of forms
  - Don't assign protocol numbers or write entities to database until process has actually "started"

SAIL - Technical Intro

- Self Assembly Interface Layers
- https://forum.appian.com/suite/wiki/73/SAIL_Components
- Used in records / tempo reports
- No GUI designer tool (yet)
  - Not needed anyway
- a! domain instead of fn! for SAIL functions
- Write it through the rules interface

- **SAIL is pure fun**

Sail Components

- Layout components
  - ColumnLayout
  - Dashboardlayout
- Link components
  - LinkField, RecordLink, etc
- Basic components
  - CheckBox, Text, Number, Dropdown, etc
- Grids
  - GridTextColumn/GridLinkColumn
- Display Fields
  - Progress Bar, Image
- Button Fields
  - Work on the same data - not very useful (yet)

Context functions!

- with / load: Create local (context) variables

```
with(
  koko: myCustomFunction(),
  koko2: anotherCustomFunction(),
  if (
    local!koko > local!koko2,
    local!koko,
    local!koko2
  )
)
```

Declare as many as you wish!

Declared variables are available in the last argument expression in the local domain

- with reevaluates the variables each time the expression is executed
- load evaluates the variables only the first time
  - For instance, use load for paging
- Domains: local! , rule! , fn! , a!, etc
  - You may drop the domain if your identifier is unique across domains
  - I recommend using it all the time to be more clear

Looping functions

- all / any / none
  - all (fn!tointeger, {1,2,3,”a”, 4}) => False
- filter / reject
  - reject (fn!tointeger, {1,2,3,”a”, 4}) => “a”
- merge
  - merge ({1,2,3}, {“a”, “b”, “c” }) => {1, “a”, 2, “b”, 3, “c” }
- reduce
  - reduce(fn!sum, 0, {1,2,3}) => sum(sum(sum(0,1), 2),3 => 6
  - the list doesn’t necessarily need to be of simple types !
- apply (==map)
  - apply(fn!tointeger, {2, “33”, “2.8}) => {2,33,3}
  - apply(fn!sum, { {1,2,3}, {3,4,5} }) =>
  - You can also pass context

```
apply(
  fn!user,
  {“a062”, “k961”},
  “firstName”
)
```

```
with(
  user1: fn!user(“a062”, _),
  apply(
    fn!user1,
    {“firstName”, “lastName”, “email”
  )
)
```

Partial Evaluation

- Replace a parameter input with a _ and you have partial function
  - user(_, “firstName”) => This is a function object that takes *one* argument and can be used to get the firstName of a user
  - Functions are first class objects

```
with(
  getfname: user(_, “firstFirstName”),
  getlname: user(_, “getLastName”),
  local!getlname(“a062”) &”, “ & local!getfname(“a062”)
)
```

- Very useful where you need to create single-argument functions
- Don’t create a new rule wherever you just need partial evaluation !!!
- _ can be named

```
with(
  user1:user(username:_, attribute:_),
  local!user1(attribute:”firstName”, username:”a062)
)
```

Starting with records

- Create entity based record
- city
  - name
  - region
    - name
    - district
      - name

Define your ListViewItem for the List View Template (title / details / timestamp / image)
** Notice the rf domain **

```
='type!ListViewItem'(
  title: rf!id& " " & rf!name,
  details: rf!region.name &", " & rf!region.district.name
)  OR USE
rule!APN_listViewTemplate(title:"foo", details:"bar")
```

A dashboard can contain two columns. Each column has fields one below the other. Notice once again the rf domain. To test that the rule works create a rule that get id, name and region name as inputs

```
=a!dashboardLayout(
  firstColumnContents: {
    a!textField(
      label: "ID",
      readOnly: true,
      value: rf!id
    ),
    a!textField(
      label: "Name",
      readOnly: true,
      value: rf!name
    ),
    a!textField(
      label: "Region",
      readOnly: true,
      value: rf!region["name"]
    )
  }
)
```

Other record options

- Show default filters
  - Very important for us - use loggedInUser()
- Show facets
  - *Must* be used only since only 100 records can be shown
- Show related actions
  - Pass parameters using the rp! domain: {cityid:rp!id}
  - Removing start form has a bad behavior
- Show searching
  - Searches in title only
    - So, record title has to contain at least our protocol number !
- Similar options with process based records
  - Please use them only for special cases (for instance cancelling processes)
  - Users shouldn’t care about the process instance - only for their data

Starting with Tempo Reports

- We don't get any help with our data in Tempo reports
  - So *we* must define it with load/with

```
=with(
  local!cities:rule!getCities(),
  a!dashboardLayout(
    firstColumnContents: {
      apply(
        a!textField(label:"city", readOnly:true, value: _    ),
        apply(
          index(_, "name", ""),
          local!cities
        )
      )
    }
  )
)
```

Define local!cities

Get the name of each city

Create list of textFields

Creating charts

```
=with(
  local!cities:rule!getCities(),
  local!regions:rule!getRegions(),
  a!dashboardLayout(
    firstColumnContents: {
      * Same as before *
    },
     ***
  )
)
```

```
secondColumnContents: {
    a!pieChartField(
      series: {
        apply(
          a!chartSeries(label: _, data: _),
          merge(
            apply(index(_, "name", ""), local!regions),
            apply(
              rule!countCitiesByRegionId,
              apply(index(_, "id", ""), local!regions)
            )
          )
        )
      },
      showDataLabels: true
    )
}
```

series is an array of chartSeries objects chartSeries also takes a link as a parameter (for instance link to record)

merge will return “F9iwtida”, 3, “Attikh”, 2, etc

Adding *non* readonly components 1

```
=load(
  local!district: null,
  local!districts: rule!getDistricts(),
  local!city: null,
  local!region: null,
  with(
      local!regions: if(isnull(local!district), "{}", rule!getRegionsByDistrictName(index(local!districts[local!district],"
name",""")) ),
      local!cities: if(or(isnull(local!region),isnull(local!district)) ,"{}", rule!getCitiesByRegionName(index(local!regions
[local!region],"name","")) ),
      local!region: if(isnull(local!district), null(), local!region ),
      local!city: if(or(isnull(local!region),isnull(local!district)) , null(), local!city),
      a!dashboardLayout(
        firstColumnContents: {
          a!dropdownFieldByIndex(
            label: "District",
            instructions: "Please select your district",
            placeholderLabel: "...",
            choiceLabels: {
              apply(
                index(_, "name", ""),
                local!districts
              )
            },
            value: local!district,
            saveInto: local!district
          ),
```

Dropdowns accept only labels

Dropdown gets value from and saves it into local variable

Adding *non* readonly components 2

```
if(local!district,
  a!dropdownFieldByIndex(
  label: "Region",
  instructions: "Please select your region",
  placeholderLabel: "...",
  choiceLabels: {
    apply(
      index(_, "name", ""),
      local!regions
    )
  },
  value: local!region,
  saveInto: local!region
),
  rule!APN_uiTextReadOnly(label: "Region",value: local!region)
),
```

Visible only if local!district is not null

Save into local variables

Adding *non* readonly components 3

```
        if(local!region,
           a!dropdownFieldByIndex(
           label: "City",
           instructions: "Please select your City",
           placeholderLabel: "...",
           choiceLabels: {
             apply(
               index(_, "name", ""),
               local!cities
             )
           },
           value: if(isnull(local!city), local!city, if(local!city > length(local!cities), null(), local!city) ),
           saveInto: local!city
         ),
           rule!APN_uiTextReadOnly(label: "City",value: local!city)
         )
       },
       secondColumnContents: {
         rule!APN_uiTextReadOnly(label: "District",value: if(isnull(local!district), "-", local!districts[local!district])),
         rule!APN_uiTextReadOnly(label: "Region",value: if(isnull(local!region), "-", local!regions[local!region])),
         rule!APN_uiTextReadOnly(label: "City",value: if(isnull(local!city), "-", if(local!city > length(local!cities), "-", local!
cities[local!city])) ),
         if(isnull(local!city), rule!APN_uiTextReadOnly(label: "City Link",value:"-"), if(local!city > length(local!cities), rule!
APN_uiTextReadOnly(label: "City Link",value:"-"),  a!recordLink(recordType: cons!SP_CITY_RECORD ,identifier: index(local!cities[local!
city], "id", 1), label:"Goto city")  ))

       }
     )

  )
```

Visible only if local!region is not null

Check if previous index was > our current cities length

2nd col for debugging. The selected city can be used for linking etc

Rule based records (ListView)

```
=load(
 customGroups:rule!getCustomGroups(),
 'type!{http://www.appian.com/ae/types/2009}ListView'(
  dataSubset: 'type!{http://www.appian.com/ae/types/2009}ListViewDataSubset'(
   startIndex:1,
   batchSize: count(customGroups),
   totalCount: count(customGroups),
   data: apply(
    rule!getCustomGroupsLVI,
    customGroups
   ),
   identifiers: apply(fn!tointeger,customGroups)
  )
 )
)
```

We create our ListView *by hand* !

```
='type!{http://www.appian.com/ae/types/2009}ListViewItem'(
 title: group(ri!g, "groupName"),
 timestamp: group(ri!g, "created"),
 details: group(ri!g, "description")
)
```

Rule based records 2 (Dashboard)

```
=load(
local!g:togroup(ri!gid),
a!formLayout(
 label:group(local!g, "groupName"),
 firstColumnContents: {
  a!textField(
   label: "Description",
   readOnly: true(),
   value:group(local!g, "description")
  ),
  rule!createUsersGridForGroup(local!g)
 }
)
)
```

```
=load(
 local!pagingInfo: a!pagingInfo(
  startIndex:1,
  batchSize: 10
 ),
with(
 local!datasubset: todatasubset(
  getgroupmemberuserspaging(
   ri!g,
   0,-1
  ), local!pagingInfo
 ),
```

```
a!gridField(
 totalCount: local!datasubset.totalCount
 columns: {
  a!gridTextColumn(
   label:"Username",
   data: local!datasubset.data,
   field: "username"
  ),
  a!gridTextColumn(
   label:"Name",
   data: apply(
    rule!getUserFullname,
    local!datasubset.data
   )
  ),
  a!gridTextColumn(
   label:"Custom Field",
   data: apply(
    user(_, "customField1"),
    local!datasubset.data
   )
  )
 },
 value: local!pagingInfo,
 saveInto: local!pagingInfo
)
```

The record dashboard also contains a gridField

Another Chart

```
a!columnChartField(
 categories : {"Custom Groups"},
 series: rule!getCustomGroupData(),
 yAxisTitle: "Αρ. χρηστών",
 showLegend: true,
 showDataLables: true
)
with(
 local!groups:rule!getNonEmptyCustomGroups(),
 apply(
  rule!getChartDataForGroup,
  local!groups
 )
)
a!chartSeries(
   label: group(ri!g,"groupName"),
   data: count(fn!getgroupmemberuserspaging(ri!g,0,-1)),
   links: a!recordLink(
   recordType: cons!CUSTOM_GROUP_RECORD_TYPE,
   identifier: group(ri!g,"id")
  )
 )
```