Important Notices
The material contained in this publication (including any supplementary information) constitutes and contains confidential and proprietary information of Infor.

By gaining access to the attached, you acknowledge and agree that the material (including any modification, translation or adaptation of the material) and all copyright, trade secrets and all other right, title and interest therein, are the sole property of Infor and that you shall not gain right, title or interest in the material (including any modification, translation or adaptation of the material) by virtue of your review thereof other than the non-exclusive right to use the material solely in connection with and the furtherance of your license and use of software made available to your company from Infor pursuant to a separate agreement, the terms of which separate agreement shall govern your use of this material and all supplemental related materials ("Purpose").

In addition, by accessing the enclosed material, you acknowledge and agree that you are required to maintain such material in strict confidence and that your use of such material is limited to the Purpose described above. Although Infor has taken due care to ensure that the material included in this publication is accurate and complete, Infor cannot warrant that the information contained in this publication is complete, does not contain typographical or other errors, or will meet your specific requirements. As such, Infor does not assume and hereby disclaims all liability, consequential or otherwise, for any loss or damage to any person or entity which is caused by or relates to errors or omissions in this publication (including any supplementary information), whether such errors or omissions result from negligence, accident or any other cause.

Without limitation, U.S. export control laws and other applicable export and import laws govern your use of this material and you will neither export or re-export, directly or indirectly, this material nor any related materials or supplemental information in violation of such laws, or use such materials for any purpose prohibited by such laws.

Do not copy or distribute without proper authorization.

Trademark Acknowledgements
The word and design marks set forth herein are trademarks and/or registered trademarks of Infor and/or related affiliates and subsidiaries. All rights reserved. All other company, product, trade or service names referenced may be registered trademarks or trademarks of their respective owners.
Contents

Introduction ........................................................................................................................................... 5
  Landmark Pattern Language (LPL) for Configurations ................................................................. 5
  What Is the Configuration Console? ............................................................................................... 5
  Configuration Console Development Process and the Landmark Pattern Language........ 5

General Definitions ............................................................................................................................ 8
  Base Definitions .............................................................................................................................. 8
  Data and Field Types ..................................................................................................................... 8
  Fields and Values .......................................................................................................................... 10
  Operators ....................................................................................................................................... 10
  Dates and TimeStamps .................................................................................................................... 12
  References ..................................................................................................................................... 13
  Actions and Actors ........................................................................................................................ 15
  Messages ........................................................................................................................................ 16
  Arrays ........................................................................................................................................... 16

Fields and Conditions ...................................................................................................................... 18
  Derived Fields ............................................................................................................................... 18
  Conditions .................................................................................................................................... 20

User Interface Definition ................................................................................................................. 21
  Base Definitions ........................................................................................................................... 22
  Drills ............................................................................................................................................... 26
  Navigations ................................................................................................................................... 26
  Context Messages ........................................................................................................................ 27
  Lists ............................................................................................................................................... 27
    Card View ................................................................................................................................. 31
    Instance Count Chart ................................................................................................................. 31

Forms ................................................................................................................................................. 31
  Base Definitions ........................................................................................................................... 32
  Action ........................................................................................................................................... 34
  Composite ..................................................................................................................................... 34
  Matrix ........................................................................................................................................... 36
  Search .......................................................................................................................................... 38
  Summary ....................................................................................................................................... 38
Page Definition...............................................................................................................38
Security Class Definition..............................................................................................42
    Structure Definitions ...............................................................................................43
    Base Definitions ......................................................................................................43
Introduction

Landmark Pattern Language (LPL) for Configurations

What Is the Configuration Console?

The Configuration Console is a tool in the Infor Rich Client that enables administrators to make several types of changes that affect Landmark applications and users.

**Application** Look-and-feel changes to menus, pages, and various user interface objects and actions associated with a business class, and configuration changes for features that apply across a data area, including

- enabling and disabling data translation (multi-language field configuration)
- creating business subjects, and
- creating and modifying MIME types.

**Security** Creation and modification of security classes

You can also use the Configuration Console for the maintenance of actor, identity, and role records.

**Web Services** Creation of new web services

Web services enable non-Landmark systems to view or update Landmark data.

When you make configuration changes through the Configuration Console, the changes apply across the application and to all users.

Configuration Console Development Process and the Landmark Pattern Language

The Landmark Pattern Language (LPL) is the language that many of the source files in Landmark applications are written in, such as those for business classes and user interface objects. The Configuration Console user interface allows manipulation of an application’s LPL. The application LPL source file integrity is fully protected by the Configuration Console; the LPL source file is not modified by the console, rather a copy of the LPL source file is created and stored to a database table as a configuration. At application execution time, this configuration is brought into memory as an overlay of the base source file and is executed in place of the original LPL source file. The key point is that configurations are dynamic and take effect in real time, therefore please use caution if using the Configuration Console on a production instance of an application.
When creating configurations with the Configuration Console, it is best to do so in a provisional or test scenario before making the configurations available to all users. If you make the configuration changes directly to a production environment, the changes are applied as soon as you save them in the Configuration Console.

Generally, you should use the Configuration Console user interface to add, delete, and modify configurations. However, it can be useful to view the LPL in the Configuration Console to better understand possible configuration changes. For example, you might want to view the LPL for your configuration and then compare it to the original LPL.

The LPL is exposed through **Edit LPL** buttons and **view base LPL** links. If you view LPL in the Configuration Console, you can use **Ctrl+F** to open a search box at the bottom of the LPL pane to search through the LPL text.

**Edit LPL button in the Configuration Console**

Although rare, it might be useful to manipulate a configuration’s LPL manually using the Edit LPL pane. The remainder of this document serves as a reference guide for understanding the LPL.

**IMPORTANT:** Manually modifying the LPL is **at your own risk**. These manual modifications are not supported by Infor.

You have a selection of tools to help you identify configuration issues. These are in addition to any error messages that you see while working on a configuration in the Configuration Console. These tools can help you identify invalid configurations, locate specific errors within configurations, and compare different versions of a configuration to better understand what might be causing a problem.

- Verify Configurations (**cdverify**) utility
  Use the utility to quickly list invalid configurations and show any syntax errors in configurations.
- Configuration Console
  The Configuration Console includes tools for comparing configurations, locating invalid syntax, and identifying invalid configurations.
• Configuration business class forms
  The Data menu in the Infor Rich Client lets you view a list of configurations of a single type, and then identify invalid ones of that type and compare any two versions of the configuration.

Please refer to the Landmark documentation *Infor Landmark Technology User Interface Guides* for detailed information on using the Configuration Console.
## General Definitions

// Scope is defined by indent level. An indent level is 4 spaces or 1 tab.

### Base Definitions

<table>
<thead>
<tr>
<th>LPLConstructName</th>
<th>::= &lt;uppercase character&gt;[&lt;alphanumeric characters&gt;...] // up to 255 characters</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActionName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>ActionTag</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>BusinessClass</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>BusinessTask</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>CardViewName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>ConditionName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>ContextMessage</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>ChartName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>CubeRelationName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>DetailSectionName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>DrillName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>DrillListName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>FieldGroupName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>FieldName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>FormInvokeName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>FormName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>FullStateName</td>
<td>::= &lt;StateName&gt;[.&lt;StateName&gt;...]</td>
</tr>
<tr>
<td>ImageMapName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>ListName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>MatrixForm</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>MenuItemName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>MenuName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>ModuleName</td>
<td>::= &lt;alphanumeric characters&gt;</td>
</tr>
<tr>
<td>NavigationName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>PageName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>PanelName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>PaneName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>PeriodViewName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>RelationName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>SecurityClassName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>SetName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>StateName</td>
<td>::= (&lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>Subject</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
<tr>
<td>WebAppName</td>
<td>::= &lt;LPLConstructName&gt;</td>
</tr>
</tbody>
</table>

### Data and Field Types

| DimensionField             | ::= <dimension RelatedField>                                                    |
| FieldSize                  | ::= <numeric characters>                                                        |
| KeyField                   | ::= <keyfield FullFieldName>                                                    |
Literal ::= <any character except whitespace or angle brackets>

NbrDecimals ::= <numeric characters>

Number ::= [-]<numeric characters>[. <numeric characters>]

# ::= <numeric characters>

Percent ::= <Number>%

Text ::= " "<alphanumeric characters> '"'

DataDefinition ::= (is a[n] <FieldName>
  | is like <FieldName>
    // Must be a simple field; this syntax determines only type and size
  | is a[n] <BusinessClass> group [in subject <Subject>]
    // Subjects defined in Product Line Definition
  | is a[n] <BusinessClass> compute [in subject <Subject>]
  | is <TypeDataDefinition>
    // needs to have a standard set of 'system' fields: product line, module, and so on
)

Parens ::= ( '{'
  | '}'
)

PrimitiveType ::= ( Alpha
  | AlphaRight
  | AlphaUpper
  | Anniversary // cannot translate
  | BinaryDocument
  | BinaryObject
  | Boolean // cannot translate
  | BusinessObjectReference [to <BusinessClass>] // cannot translate
  | Date // cannot translate
  | [Unsigned] Decimal // cannot translate
  | DocumentTitle
  | EmailAddressField [with multiple addresses]
  | GroupField // cannot translate
  | Integer // cannot translate
  | Iteration of <BusinessClass> // cannot translate
  | MimeType // cannot translate
  | Numeric // cannot translate
  | Password // cannot translate
  | [Unsigned] Percent // cannot translate
  | Period // cannot translate
  | RichText
  | Text
  | Time // cannot translate
  | TimeStamp // cannot translate
  | UniqueID // cannot translate
  | URI
  | URL
  | Year // cannot translate
)

TypeDataDefinition ::= [<PrimitiveType> [[size (fixed | up to)] <FieldSize>[.<NbrDecimals>]]

  // 'up to' means this is a variable size field (for example, varchar)
  // 'up to' on LOB types will restrict the size of the LOB; it is useful in some DBs to reduce the base record size
  // 'fixed' is the default, however a warning is generated on any non up to type that is 30 spaces or more
  // unless fixed is designated.
  // valid on Alpha and AlphaUpper only
  // FieldSize is the overall size of the field. A Decimal field of size 12.2 means 12 total digits
  // of which 2 are the number of digits after the decimal
Fields and Values

**BusinessObjectReference** ::= <FieldName> // Field that is of type BusinessObjectReference

**DefaultLabel** ::= default label is (<LiteralMessage> | untranslatable)
   // 'untranslatable' marks the base translation text as untranslatable
   // This overrides the default base label translation text. The default base label is the LPLConstructName with spaces inserted
   // before each uppercase character.

**FieldOrArrayName** ::= ( <FieldName> | <ArrayComponent> )

**FullFieldName** ::= ( <FieldOrArrayName>[.<FieldOrArrayName>...] | <CreateStamp> | <CreateDate> | <UpdateStamp> | <RelevanceScore> | <ActionAttribute> | <BODId> | <CurrentAsyncId> has future changes // Boolean field: returns true if future-dated records exist
   // references the set of user fields that have been defined in this context or the explicitly given BusinessClass context
   user fields[<Parens><BusinessClass><Parens>]

**BODId** ::= bod id // references the BODId GroupField if this Business Class implements BODId

**CurrentAsyncId** ::= current async action request id // valid only when in a background action

**JavaFormat** ::= ""(<Literal> | <"<RelatedValue>"')..."" // Literal cannot have spaces or angle brackets
   // Must resolve to a JavaFormat as defined here:
   // http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html

**RelevanceScore** ::= relevance score // returns the relevance score from a text search – it is valid only on a List Definition

**States** ::= States
   <StateName> value is <Literal> [with <Icon>] [<DefaultLabel>]

**Value** ::= ( <Text> | <Number> | <#> [year[s] | month[s] | day[s]] | <Percent> | <Constant> | <ActionTagValue> )

Operators

**AsOfDate** ::= ( as of (<RelatedValue> | all dates)
   | after <RelatedValue>
   | before <RelatedValue>
   | between <RelatedValue> and <RelatedValue>
General Definitions

// RelatedValue must be a Date or TimeStamp

**AsOfOperator ::=** `<Parens><AsOfDate><Parens>`

**Constant ::=**

( true
| false
| blank
| high value
)

**CreateOrUpdateMode ::=** mode

// valid only in the title of a Form; results in "Create" or "Update"

**FieldOperator ::=**

( not
  // valid only with a Condition
  |
  !
  // "
  |
  no
  // "
  |
  old
  // valid only with multi-valued (array, otm) RelatedField or RelatedCondition
  |
  any
  // "
  |
  all
  // "
  |
  first
  // "
  |
  last
  // "
  |
  sum
  // "
  |
  avg
  // "
  |
  min
  // "
  |
  max
  // "
  |
  floor
  |
  ceiling
  |
  pending
  |
  instance count of
  // "
  |
  sizeofarray
  |
  (first | last) iteration of
  // valid on Iteration fields only
  |
  next
  // "
  |
  previous
  // "
  |
  reference to
  // valid only with a OTO RelatedLink only – returns the BusinessObjectReference of the RelatedLink
  |
)

**LinkOperator ::=**

( first
| last
)

**PeriodOperator ::=**

( [(prior | next) year [<#>]] [(prior | next)] {period[s] | week[s] | month[s] | quarter[s] | [all] year[s]} [<#> [thru <#>]]
| [(prior | next) year [<#>]] [(prior | next)] {period | week | month | quarter | year} [<#>]
| (beginning | ending) [ytd] balance
)

**PeriodLabel ::=**

period label

// valid only in the context of a period within a CubeView

**TotalOperator ::=**

( running total
| percent of total)

**TypeOperator ::=**

( decimals
  // valid with a Decimal type field only
  |
  size
  // size of the 'string' value of a field
  |
  year
  // valid with Date and TimeStamp fields only
  |
  month
  // " – 1-based month number (1-12)
  |
  month [short] name
  |
  day
  // " – 1-based day number (1-31)
  |
  day [short] name
|
General Definitions

| week day          // “– 1-based week day number (1-7) |
| year day          // “– 1-based year day number (1-366) |
| week              // “– 1-based week day number (1-52) |
| week year         // “– year corresponding to week operator (12/31/15) = week 1, week year 16 |
| days in month     // “– number of days in the date’s month (28-31) |
| days in year      // “– number of days in the date’s year (either 365 or 366) |
| as years          // valid with Numeric type field only – treats value as number of years |
| as months         // valid with Numeric type field only – treats value as number of months |
| as days           // valid with Numeric type field only – treats value as number of days |
| hours             // valid with Time type fields only – Decimal total number of hours since day beginning |
| minutes           // “– Decimal total number of minutes since beginning of day |
| seconds           // “– Decimal total number of seconds since beginning of day |
| date              // valid with TimeStamp type fields only (returns Date in GMT time zone) |
| system date       // valid with TimeStamp type fields only (returns Date in system’s default time zone) |
| time              // valid with TimeStamp type fields only |
| anniversary       // valid with Date and TimeStamp type fields only |
| period            // valid with Date and TimeStamp type fields only |
| levels            // valid with array fields only. Defines number of entries up to and including last entered value |
| entries           // valid with array fields only. Defines number of entered fields in the array |
| compact format    // valid only with a field that implements CompactFormat |
| version           // valid only with a key field that implements Versioning; will address the version field |
| label             // returns the translated label for the field |
| text              // valid only with a field that has Text Variable; retrieves the text with the variables replaced |
| mime type         // valid with BinaryDocument and BinaryObject only |
| translations      // valid on right side of assignment operator only |
| document [for <RelatedLink>] // valid only with a field that is a ‘document template’. This will retrieve the fully replaced document if no related link is specified then it is presumed to be based on ‘this instance’ |
| compute value [for <RelatedLink>] // if RelatedLink is an action request id, document will be replaced based on the action request |
| as of <RelatedValue> // related value must be a Date or TimeStamp |
| (date | audit entry) (first | last | next) changed [(from | to) <RelatedValue>] |
| days was <RelatedValue> [while <Condition>] [between <RelatedValue> and <RelatedValue>] |
| weighted average [between <RelatedValue> and <RelatedValue>] |
| <PeriodOperator> |
| using <JavaFormat> |
| cube dimension value [using year of <RelatedValue>] |
| input value       // always returns the input value on a transient field with a derived value |

Dates and TimeStamps

**CreateDate** ::= create date  // returns the most recent create date from the current ‘as of’ date

**CreateEffectiveDate** ::= create effective date  // returns the most recent create effective date from the current ‘as of’ date

**CreateStamp** ::= create stamp[.actor]  // references the CreateStamp in the business class |
// by itself it is the actual TimeStamp when the instance was created |
// the actor keyword references the actor who created this instance
CurrentDateTime ::= ( current time
| [system] current date
| [system] current year
| [system] current period
| [system] current timestamp
| [system] current anniversary )

DurationVars ::= ( duration begin date
| duration end date )

UpdateStamp ::= update stamp[.actor] // references the UpdateStamp in the business class
// by itself it is the actual TimeStamp when the instance was last updated
// the actor keyword references the actor who last updated this instance

References

BaseURL ::= base url<Parens>webapp is <WebAppName><Parens> // references the current base url context

BusinessClassTotal ::= total<Parens><BusinessClass>..<RelatedValue>[..<DimensionField> = <RelatedValue>]..<Parens>
// first RelatedValue must be a defined TotalName on the specified BusinessClass
// DimensionField is a dimension in the BusinessClass

ConfigVar ::= config.<Literal>
// Allows reference to a configuration variable. Configuration variables are defined in the BusinessClass
// ConfigurationParameter. If it is not found it is blank.

DataLink ::= [<LinkOperator>]  // valid within a Relation Definition
  ( <KeyField> [set]  // valid within an invocation
    | <hierarchy KeyField> ( parent
      | [and] children
      | [and] siblings
      | [and] descendants
      | [and] ancestors
      | descendant
    )

    | <RelationName>
    | <CubeRelationName>
    | <BusinessClass>[<Parens><KeyField> | <BusinessObjectReference><Parens>]<Parens>] [set]
    | <BusinessObjectReference>[<Parens><BusinessClass><Parens>]
    | <BusinessClassTotal>
    | <Agent>
    | <SessionKey>
    | actor.context.<KeyField> // retrieves the set of past and future (effective-dated) records
    | audit log records
    | [(draft | in process | completed | rejected)] [<BusinessClass>.Create] action requests
    | this instance
    | related
    | each[<Parens><LPLConstructName><Parens>]] // valid within a for each loop
    | invoked
    | result
    | child
    | cube[<Parens><BusinessClass><Parens>]] // used to address the AnalyticCube business class

AsOfOperator

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>

locale of <RelatedValue><Parens>
**General Definitions**

**SessionKey** ::= `session.key.<FieldName>`

// Allows reference to a session key. Session keys are high level KeyField values typically tied to the
// invocation of a WebApp by specifying them on the url (e.g. 'session.key.SupplierGroup=8181')

**URLVar** ::= `url.<Literal>`

// Allows reference to a url variable. These variables are available when the flag 'byw' is set to true on the url
// (…&byw=true& - automatically done on a linkback) or when they are individually specified in the WebApp
// definition as a URL Parameter

**Actions and Actors**

**ActionAttribute** ::= ( `<Actor>`
  | action
  | action comment
  | action type.[.(Create | Update | Delete | Unknown)]
  | action tag
  | applied stamp
  | audit entry id
  | audit period.[.(Past | Current | Future)]
  | correction
  | correction comment
  | effective date
  | effective time zone
  | effective through
  | entry stamp
  | initiating action
  | invoking action
  | reason code
  | subject
  | system stamp
  | action request id
  | changed field names

) // action attributes either reference the currently executed action or an audited action
// an audited action can be addressed using 'for each audit log records'
// within this for each, the 'each' keyword can be used in conjunction with these attributes.
// for example, ‘each.action’ will return the name of the past (or future) action
// 'each.effective date' will return the effective date of that past action

**ActionTagValue** ::= action tag.<ActionTag>

// allows reference to a specific action tag value, which are defined in the Product Line Definition file

**Actor** ::= [(authenticated | agent)] actor.[.(<ActorAttribute> | context.<FieldName>)]

// 'actor' by itself is the current actor which is represented by the Actor KeyField in the environment product line
// 'authenticated actor' is useful in an Actor Proxy situation
// 'agent actor' is the actor that this Agent Stereotype is linked to
// 'context.<FieldName>.' is any KeyField within this product line – actors can have any keyfield
// within the system defined as a context field with a specific value (e.g. Company = 1 for actor ApClerk1)
General Definitions

ActorAttribute ::= ( <actor FieldName> 
| initiator 
| approver 
| final approver ) 

Agent ::= [[authenticated] actor.]agent[<Parens><BusinessClass><Parens>] 

LinkBack ::= linkback<Parens> 

Messages

IsoLocale ::= <Literal> 

LiteralMessage ::= ‘”’ ( <Literal> | ‘’<RelatedValue>’’ ) ’”’ 

Message ::= [(untranslatable: | configuration:)]<LiteralMessage> 

MessageTranslation ::= translation for <IsoLocale> is <LiteralMessage> 

Arrays

ArrayComponent ::= ( <array RelatedField> 
| <array RelationName> 
| <ArrayRangeAddress> 
| <special index SingleValueArrayAddress> ) 

// if this is used when doing a ‘for each audit log records’ then it refers to the actor of the particular audit log action

// Field on Actor.busclass in environment product line
// returns true if the actor is the initiator of a RequestAction
// returns true if the actor is an approver of a RequestAction
// returns true if the actor is a ‘final approver’ of a RequestAction

// If this business class is an Agent, another agent that has been linked to this agent can be addressed with the // 'agent(<BusinessClass>)' designation. To access agents that the current actor has been linked with, add the // 'actor' keyword in front of the 'agent' keyword.

// The linkback statement is a single line statement. These options have been placed on separate lines in the BNF for readability.
// Suppresses display of the header and navigation bar
// This will result in a full http url link that will bring up the designated navigation. If a 'text is' // message is defined, then the linkback will be an HTML link, otherwise it will be a simple url.

// must be a valid IsoLocale
// Literal cannot have spaces or angle brackets
// 'CompanyIsRequired' will display the message 'Company is required'
// 'Company<Company>IsRequired' will display the message 'Company 123 is required'

// valid only when 'configuration:' used
// 'untranslatable' is valid in all LPL
// 'configuration' valid in a configuration only

// This can also be a simple Alpha field which then implicitly addresses each character
// being mapped to the index value is an array (this is done using the each operator). A Field becomes an array when it is an ArrayField or when it is a derived field that contains an ArrayField without a single subscript denoting a specific occurrence. The RelatedField expression (A + each B) where B is an ArrayField of size 10 returns an array of size 10 where A is added to each occurrence of B. More complex array expressions are defined below. An ArrayRangeAddress is a subset of an array that can result in a single occurrence or even no occurrence. A special index SingleValueArrayAddress is an array // with a specific special index variable associated with it. By default all ArrayComponents are associated with // the special index variable i except when two arrays are used in an expression where it is ambiguous as to // how to associate their respective occurrences. For example, (Array1 * Array2), is the result a single
General Definitions

ArrayRangeAddress ::= ArrayComponent \[ \langle I \rangle = \langle from \ IndexVariable \rangle \text{ to } \langle to \ IndexVariable \rangle \]  
// This expression defines a subset range of the ArrayComponent denoted by two index variables: the  
// from IndexVariable and the to IndexVariable. The to IndexVariable is inclusive. The size of this array  
// is (to IndexVariable – from IndexVariable) + 1.

I ::= ( i | j | k )  
// These are internal array indexing variables, which are for special use only within array expressions.  
// Their purpose is to allow the explicit mapping of multiple array occurrences in a single expression.  
// The i variable is implicitly used and is accessible in any array expression; for example, for each  
// <ArrayComponent> is equivalent to for each <ArrayComponent> \[i=1:arraysize\]; so the  
// special index variable i can be used to explicitly address an occurrence within the for loop. This can be  
// useful for comparing the current occurrence with the next occurrence: if (A[i] == A[i+1]).  
// Any Array indexed with a special index is still considered an ArrayComponent as it implicitly means  
// all occurrences in the absence of any array occurrence context.

IndexVariable ::= ( <RelatedValue> 
| <I> 
| <SizeOfArray> )  
// Arrays are indexed beginning with 1. The first occurrence of an array is denoted with an index of 1.  
// An index variable can be a RelatedValue that is not an array or the special index variables i, j, and k.

SingleValueArrayAddress ::= ArrayComponent \[ \langle IndexVariable \rangle \]  
// A specific value in an array is addressed by using hard brackets [ ] to denote the  
// index of the specific occurrence. The special index variables only have meaning when used within  
// an array context that is an ArrayExpression or a for each <ArrayComponent> scoped statement.

SizeOfArray ::= sizeofarray ArrayComponent  
// sizeofarray returns the size of the first array component in the ArrayComponent (an ArrayComponent  
// can be made up of several ArrayComponents).
Fields and Conditions

Derived Fields

<FieldName> is a <DerivedFieldType>
[<DefaultLabel>] [restricted]
[<DerivedFieldExpression> | <optionally blank if NativeField>]

DerivedFieldType ::= 
  { ConditionalField
    | ComputeField

Examples of Array Expressions

A = ArrayField[5]
A = ArrayRelation[i].FieldName

A = sum ArrayField
   which is equivalent to
A = sum ArrayField[1:sizeofarray]
   which is also equivalent to
A = sum (ArrayField[j][j-1:sizeofarray ArrayField]

A = sum ArrayRelation.FieldName

Searches each relation and returns true if it finds a match.
If (FieldName == any ArrayRelation.FieldName)

Multiplies each occurrence of Array1 with the respective occurrence of Array2
returning a single dimension array of the same size as Array2 and then sums up all the values.
A = sum (Array1[i] * Array2[i][i-1:sizeofarray Array2]

Shifts the array 1 to the left leaving the last value the same.
Array1 = Array1[i+1][i-1:sizeofarray-1]

Shifts the array 1 to the left and fills the last value with blank.
This is because an out of bounds array access results in the blank value.
Array1 = Array1[i+1][i-1:sizeofarray]

A = (ScalarValue + each ArrayValue)
   which is equivalent to
A = (ScalarValue + ArrayValue[i][i-1:sizeofarray])
   which is also equivalent to
A = (ScalarValue + each ArrayValue[i][i-1:sizeofarray])

The result of this is a single dimension array the size of ArrayValue where each occurrence of ArrayValue has ScalarValue added to it.
DerivedFieldExpression ::= 

  ConditionalField  
  <ConditionalFieldControl>

  ComputeField  
  <Parentheses><RelatedValue> [<MathOperator> <RelatedValue>]<Parentheses>...

  StringField  
  <RelatedValue> <RelatedValue> <RelatedValue>...

  MessageField  
  <Message>  // language translatable string that is treated as a Message (lowercase)

  LabelField  
  <Message>  // language translatable string that is treated as a Label (upper and lowercase)

  DerivedField  
  (<Rule>...)

Condition ::=  
  <Parentheses><ConditionNode><Parentheses>  
  [<Conjunction> <Parentheses><ConditionNode><Parentheses>...]

ConditionalFieldControl ::= 
  if <Condition>  
  (<RelatedValue> | <ConditionalFieldControl>)...  
  [else]  
  (<RelatedValue> | <ConditionalFieldControl>)...  

MathOperator ::=  
  { +  // You can use the ‘+’ operator to concatenate Alpha type fields  
  | −  // You can use the ‘-’ operator to remove strings from Alpha type fields  
  | *  | /  | ^  | % }

Rule ::= (return [<RelatedValue>] | <ControlRule>)

ControlRule ::=                     // A ControlRule is a rule that defines the control of other rules. As with a standard rule, each has a recursive  
  // nature; that is, the <Rule> under the ‘If <Condition>’ can be another ControlRule or standard rule.,  
  if <Condition>  
  <Rule>...  
  [else]  
  <Rule>...  

for each[<Parentheses><LPLConstructName><Parentheses>] (  
  [<Distinct> in] <otm RelatedLink>  
  | <array RelatedField>  
  | <iteration FullFieldName>  
)

[while <Condition>]  
  <Rule>...  

[each[<Parentheses><LPLConstructName><Parentheses>]..<RelatedField>]  
  // Within the scope of the ‘for each’ the fields of the <OTM RelationName> and the occurring field within  
  // <Array RelatedField> are addressed through the ‘each’ key word. If the <OTM RelationName> is 'Employees'  
  // and a field on the Employee business class is 'Name' then that field is addressed as 'each.Name' within the scope  
  // of the for each. If one ‘for each’ is nested within another ‘for each’ a ‘for each I’ can be specified that allows both
// the for each and the for each1 variables to be addressed through each.<Field> and each1.<Field>
// If this is for an array field, this structure drives a loop that goes through each occurrence in the array.
// All other array fields are presumed to have the same index as the primary occurring field. The RelatedField must be a field
// that has a Representation with a field that occurs. The actual occurring field must be addressed directly (for example
// each.OccurringField) within the actual loop.
// If the ‘each’ qualifier is not provided it will use the most inner for each context.
// If an iteration field is used it must first be initialized by doing a “first iteration of...”
end for each

// Used to break out of a ‘for each’ loop. This rule is NOT a syntactical structure to end a ‘for each’ statement. Rather, it is
// a control rule ending the execution of the ‘for each’ rule and causing execution to go to the first rule after the ‘for each’.
while <Condition> <Rule>...
end while

// Used to break out of a ‘while’ loop. This rule is NOT a syntactical structure to end a ‘while’ statement. Rather, it is
// a control rule ending the execution of the ‘while’ rule and causing execution to go to the first rule after the ‘while’.

Conditions

<ConditionName>
[<DefaultLabel>]
[restricted]
when <Condition>

ConditionOperator ::= ( =
| !=
| >=
| <=
| >
| <
| !>
| !<
| like
| contains
| [not] within
| [not] overlaps
| [not] matches
)

// The like operator is similar to the ‘=’ operator except that it will do wild-card searching
// The contains operator will do a text search on the field; it is valid only if the field is text searchable
// The within operator is valid with array key fields that implement the Hierarchy pattern
// It evaluates to true when the left operand’s occurring fields match up to the non-blank right
// operand’s occurring fields. If the left operand has {5,3,9,10,20} as its occurring values it will match
// the following right operands: {5,,}, {5,3,,}, {5,3,9,,}, {5,3,9,10,}, {5,3,9,10,20}. The concept is that
// the left operand is ‘within’ the right operand’s hierarchical structure.
// The within operator is also valid with group fields that implement the FieldRange pattern
// The overlaps operator is valid only with group fields that implement the FieldRange pattern
// The matches operator treats the right side operand as a regular expression to determine if the left side operand
// completely matches the pattern on the right. The java doc for java.util.regex.Pattern contains some
// documentation for a number of the various regular expression strings. Another resources is

Conjunction ::= ( and
| or
)

ConditionNode ::= ( <ActorCondition>
| <ChangedCondition>
User Interface Definition

// UI definitions are logically part of either a business class, business task, or field definition. This is accomplished by having the same file name as the // respective business class, business task, or field in a different directory structure. The standard directory structure is com/lawson/apps/<ModuleName>; // the UI definition files are stored in com/lawson/forms/<ModuleName>. Thus the file com/lawson/forms/field/Address.field contains the UI definitions for // the field com/lawson/apps/field/Address.field. Every business class, business task, or field can have a default UI definition as well as alternate UI definitions.

// If no Drill List is specified then the Drill List is automatically built from all non-drill restricted Navigations and all Lists and Forms that are // propagated as a Drill. If a List is propagated as a Drill then it does not put a drill on this business class but rather on all the business classes that // are in the context of this business class. Thus it may make sense to have a Drill List specified in this business class as well as lists that are // propagated as a Drill. However, it would not make sense to propagate a form as a Drill or to drill restrict any Navigations if a Drill List is specified.
// as they would not be automatically put in the Drill List.

UI Structure ::= 

{ <BusinessClass> is a BusinessClass
  | <BusinessTask> is a BusinessTask
  | <FieldName> is a Field
}

[Drill List]
  [<DrillDefinition>…]
  [<DrillListDefinition>…]
  [<NavigationDefinition>…]
  [<ContextMessageDefinition>…]
  [Context Message Invocations]
    [<ContextMessageInvocation>…]
  [<ListDefinition>…]
  [<CardViewDefinition>…]
  [<InstanceCountChartDefinition>…]
  [<FormDefinition>…]
  [<ActionFormDefinition>…]
  [<CompositeFormDefinition>…]
  [<MatrixFormDefinition>…]
  [<SearchFormDefinition>…]
  [<SummaryFormDefinition>…]

Base Definitions

Action ::= action is [<StateName>.,<ActionName>…]

ActionIcon ::= ( apply
  | approve
  | award
  | calculate
  | chart
  | claim
  | copy
  | create
  | delete
  | edit
  | export
  | finalize
  | open
  | pdf
  | preview
  | print
  | process
  | reject
  | release
  | request
  | save
  | search
  | submit
  | transfer
  | undo
  | validate

Apply Preview
Approve Pdf
Award Print
Calculate Process
Chart Reject
Claim Release
Copy Request
Create Save
Delete Search
Edit Submit
Export Transfer
Finalize Undo
Open Validate
AlertType ::= ( red        // alerts shown in order of precedence
| yellow
| green
| blue        // blue is displayed only if no higher alert condition is met
| <Icon>      // displayed only if no higher alert condition is met
)

CalledOutAction ::= ([action is] ([<RelatedLink>..<StateName>..<ActionName> | <StandardAction>
| navigation is <NavigationName>
| link is <UILink>
)
  [action icon is <ActionIcon>]
  [label is <Message>]
  [mouse over text is <Message>]
  [hide when invalid]
  [valid when <SimpleCondition>]

CheckControl ::= check control
  {[valid | visible) when <SimpleCondition>]
  [checked state is <FieldName>]
  [check action is [<StateName>..<ActionName>]|<StandardAction>]
    // optional for printing purposes
  [when <SimpleCondition>]
    // must have a condition if defining multiple targets
  [uncheck action is [<StateName>..<ActionName>]|<StandardAction>]
    // must have a condition if defining multiple targets
  [<Label>]
    // default is no label
  [display as switch]
  {[left align | center | right align]}
    // left align is default unless used in a MatrixForm (center is default)
  {[top align | middle | bottom align]}
    // top align is default
  [align as label]
  [indent]
  [hidden]
  [when value changed]
    ( <RelatedField> = <RelatedValue> )
      // RelatedField must be a mutable field on a form or list
    [refresh <RelatedField>]
      // RelatedField must be on a form or list
  )

Color ::= { black
| blue
| brown
| gray
| green
| jade
| magenta
| maroon
| orange
| red
| yellow
}

DisplayType ::= { browser
| desktop
| smartphone
| tablet
}
Form ::= [independent] form is [<RelatedLink>].(<FormName> | primary | inline)

FormText ::= 
( text of (<Message> [with <Icon>] | <Icon>)
 [TextStyle]...) 
[link is <UILink>] 
[align as label] // make text line up in label column
| blank line 
)

Header ::= 
header(1|2|3|4|5) of <Message>
 [{(show border | underline) [color of <Color>]}
 [(shade]
 [(center | right align)]

HttpURL ::= <Message>

Icon ::= icon.<IconName>

IconName ::= <Literal> // IconName is a dynamic keyword that comes about by having either a .png file or a .svg file in the directory
 // <LASRCDIR>/<ProductLine>/icons. The .png files can have the following format: ’<ImageName>_<Size>.png’
 // where <Size> is a number denoting the pixel size of the icon, for example: 16, 32, 72. All files in this format
 // are considered icons and must be square.
 // The .svg files must have the format ‘<ImageName>.svg’. The .png files can also have this format.

ImageName ::= <Literal> // ImageName is a dynamic keyword that comes about by having either a .png file or a .svg file in the directory
 // <LASRCDIR>/<ProductLine>/images’.

Label ::= 
label is (<Message> | default)
 [TextStyle]...

List ::= 
list is <RelatedLink>,(<ListName> | primary [audit list])
 [search <FullFieldName> using <RelatedValue>...] 
[cannot be empty] // valid only when used in a Wizard
 [when <SimpleCondition>]
 [Message] 
[confirmation required on empty] // valid only when used in a Wizard
 [when <SimpleCondition>]
 [Message] 
[form is ((<FormName> | primary | inline)]...
 [FormDefinition] // valid only on an inline Form; this is a form on RelatedLink
 // Will cause the form to display beneath the list.
 [current period is <RelatedValue>] // RelatedValue is in context of this business class
 [current year is <RelatedValue>] // RelatedValue is in context of this business class
 [Row Dimensions] // valid only on a CubeView; overrides the CubeView Row Dimension
 <DimensionField>...
 [helper list is <RelatedLink>,(<ListName> | primary)]...
 [valid when <SimpleCondition>]
 [search <FullFieldName> using <RelatedValue>...] 
 [Action] // A ‘helper list’ is a list that will show below the main list that should be helpful to managing the main list.
 // One example of a list being helpful is a list of instances that can be copied into the main list. Another example
 // may be a list of instances that are just useful to see when managing the main list.
 // If no action is specified on a helper list then all the Instance actions will display. Multiple actions can be specified.
 // By default the parameters of the instance action will be filled by name using the context
// fields of this business class. This mapping can be explicitly defined using the 'invoked.' Syntax
// The 'RelatedValue' on the right side of the invoked syntax is a RelatedValue from the perspective of this business class.
// 'parameter' causes this field to be put on the helper list itself as an inputable field that then gets passed to the action.

Navigation ::= navigation is <NavigationName>

Page ::= page is <PageName>[.<PanelName>]

SimpleCondition ::= <Parens> ( <RelatedField> // <RelatedField> must result in a Boolean type
    | <RelatedField>.<StateName>
    | <FieldCondition> // only 'entered' and 'changed' allowed
    ) <Parens>

StandardAction ::= ( open
    | save [and (new | close)]
    | export to (pdf | csv)
    | search
    | refresh
    | charts
    | helper list
    | next record
    | previous record
    )

StatusIcon ::= ( approved
    | attached
    | complete
    | private
    | public
    | rejected
    )

Template ::= <RelatedLink>

    business class is <BusinessClass> // The parser accepts this line and the one that follows at this level of indent
    ( <Form>
    | [<FormDefinition>]
    | list is (<ListName> | primary)
    ) // A Template specification is a means for building a list of <RelatedLink> instances that are used as a template for
    // creating <BusinessClass> instances. <BusinessClass> must implement the TemplateDriven pattern. The
    // <RelatedLink> target must be the business class specified in the TemplateDriven pattern.

TextStyle ::= ( header(1|2|3|4|5) // Text Styles do not affect fields in List or AuditList definitions
    | (left align | center | right align) // left align is default
    | (top align  | middle | bottom align) // top align is default
    | bold
    | italics
    | bullet
    | left of field
    | right of field
    | color of <Color>
    )

UILink ::= ( <NavigationName>
    | [[{embedded | external | dialogue}] <HttpURL> // by default this will replace the entire frame
    | // 'embedded' will open this url within the application frame
    | // 'external' will open this url in an external browser (if in RichClient) or on another tab
    | // 'dialogue' is valid only when in a browser and will mimic a dialogue
    )
User Interface Definition

Drills

DrillDefinition ::=<DrillName> [is a[n] <NavigationName>]
   // if [is a <NavigationName>] is specified or if <DrillName> is itself a NavigationName
   // then the definition of the Navigation cannot be overridden. The Drill will simply take on
   // the definition of the Navigation.
   [valid when <SimpleCondition>]
   [(<Page> | <Form>)
      [restrict action [<StateName>.]<ActionName>…]
      [<FormDefinition>]
   ]
   [<List>]
   [position to <RelatedLink>] // <RelatedLink> must be a record within the specified List
   [<Template>]
   [view <RelatedField>] // must be a BinaryDocument or a BusinessObjectReference
   // If RelatedField is a BusinessObjectReference, the primary form or set of valid drill target forms is displayed.
   ]

DrillListDefinition ::=<DrillListName> is a DrillList
   (<DrillDefinition> | primary audit list)...

Navigations

NavigationDefinition ::=<NavigationName> is a Navigation
   // prevents this navigation from being put in the automated drill list
   [drill restricted]
   [valid when <SimpleCondition>]
   [show as pdf [in foreground]]
   // By default, navigating to a PDF causes the PDF to be generated (and stored) in the background asynchronously
   // and run through the document node, which limits how many can run at one time. This is appropriate when the PDF
   // is a longer running report; however, when the PDF is not really a long report but more like a page, it is more
   // appropriate to run it synchronously in the foreground.
   ( [<DisplayType>] <Page>
   | [<DisplayType>] <Form>
      [open action is <ActionName>]
      [restrict action [<StateName>.]<ActionName>…]
      [<FormDefinition>]
   ]
   | [<DisplayType>] <List>
      [position to <RelatedLink>] // <RelatedLink> must be a record within the specified List
   | [<DisplayType>] <Template> // must be a BinaryDocument or a BusinessObjectReference
   | view <RelatedField> // If RelatedField is a BusinessObjectReference, the primary form or set of valid drill target forms is displayed.
   )...
Context Messages

ContextMessageDefinition ::= 
   <ContextMessage> is a[n] <InforContextMessage> Message
   Property Mapping
   <MessageProperty> = <RelatedValue>...
   
InforContextMessage ::= <Literal>
   // The list of available context messages and their properties are defined here at
   
MessageProperty ::= <Literal>
   
ContextMessageInvocation ::= // valid only on a BusinessClass definition
   send [<RelatedLink>][<FullFieldName>..]<ContextMessage>
   [on (all lists | all forms | <ListName> list | <FormName> form)]...

Lists

ListDefinition ::= 
   <ListName> is (a List | a[n] <ListName> List | an AuditList)
   [is primary]
   [is drill target [for <FullFieldName>]]...
   [propagate as a drill]
   [title is <Message>] // defaults to ListName
   [show <#> lines]
   [( search form is (<FormName> | inline)
     | search field is <FullFieldName>)]
   [use as filter]
   [always display]
   <FormDefinition> // valid only on inline Form
   [implements InlineCreate]
   [implements FixedSizeList]
   [show <#> lines] // default is 10 lines
   [implements RepresentativeImageView] // business class must have a representative image defined
   [implements TreeView]
   [node icon is <RelatedField>] // must be a StateField with icon definition
   [max levels is <RelatedValue>]
   [show <RelatedValue> levels]
   [has children when <SimpleCondition>]
   [disable create within [when <SimpleCondition>]]
   [implements OrgChartView]
   [card view is <CardViewName>]
   [legend state is <RelatedField>]
   [has children when <SimpleCondition>]
   [disable create within [when <SimpleCondition>]]
   [implements CubeView]
   [detail section is row[<Parens><SetName><Parens>].<ListName>]]
   [suppress blank rows]
   [suppress current period] // by default the CubeView is in the context of the current period
   [show total node]
   [label is <Message>]
   [show <RelatedValue> levels]
   [Context]
<FieldName>…
  // Must be a dimension field. Values for this can come from a context session key, an "actor.agent(BusClass).FieldName" lookup, or from the current context before some Navigation brings up this list

Row Dimensions
<DimensionField>…
  [label is <Message>]
  [is fixed] // fixed dimensions must start at the top and be contiguous
[Period Views]
<PeriodViewName>…
  [label is <Message>]
  // valid only if the cube has a period
  ( all periods in (week | month | quarter | year)
  | Periods
    <PeriodOperator>… // if there is a PeriodOperator on a DisplayField the combination of
    [label is <Message>] // PeriodOperators must itself be a valid PeriodOperator
  )
[View Fields] // default is all the ListDisplayFields in this ListDefinition
<ListDisplayField reference>… //reference is the full text of the first line of a ListDisplayField definition

[implements DashboardView] // valid only when using a CardView (without Display Fields) and Detail Sections
[implements HorizontalScrolling]
[freeze first [<RelatedValue>] column[s]]

[auto refresh using <RelatedField>] // must be a TimeStamp field
[raise <AlertType> alert]
[mouse over text is <Message>]
[is default select]
[show in descending order]
[show in relevance order]
[suppress sub-totals]
[display negative amounts using parens]
[display amounts in (thousands | millions)]
[show in title]
[disable select in webui]

[card view is <CardViewName>] // if using both ‘card view is … ‘ and Display Fields, the card view will display in the first column
[row separator is (horizontal rule | blank line)] // default is no separator
[column header is <Message>] // valid only when using Display Fields
[sort order is (<SetName> | primary)]…
  [is default [descending]]
  label is <Message>

[Context Field Values]
<FullFieldName> value is <RelatedValue>…

[Display Fields]
( <ListDisplayField>
  | group label is <Message>
  <ListDisplayField>…
)…

[Summary Total Fields]
<FormField>…

[Instance Selection]
where <Condition>
A record in a list can be copied to a folder. This allows for related records to be copied to a folder as well.

Form to use on an 'Open'. Primary form will be used by default.

Form to use on an 'Open'. Primary form will be used by default.

Valid only on inline Form

Valid only on AuditList

Valid only on AuditList

// by default standard toolbar will show unless CalledOutActions are defined

[Additional Folder Items]

<KeyField>...

[FormDefinition]

<FormField> | <FormButton> | <CheckControl> | <FormText>...

[FormName] | inline

[drill list is <DrillListName>]

[create action is [<StateName>.]<ActionName>...]

[opan action is ((<StateName>.)<ActionName> | restricted)...]

by default a list item can always be ‘opened’; the only way to prevent this is to use the ‘restricted’ keyword

[restrict action [<StateName>.]<ActionName>...] // not valid on AuditList

[when <Condition>]

[restrict action (AuditCreate | AuditUpdate)...] // valid only on AuditList

[suppress standard toolbar] // by default standard toolbar will show unless CalledOutActions are defined

[Actions]

<CalledOutAction>...

[Quick Entry Fields]

(<FormField> | <FormButton> | <CheckControl> | <FormText>)...

[Detail Fields [(Below | On Side)]] // default is ‘Below’

(<FormField> | <FormButton> | <CheckControl> | <FormText>)...

[(valid | visible) when <SimpleCondition>]

[Detail Sections]

( <DetailSectionName> is a DashboardView

[mouse over text is <Message>]

[title is <Message>]

[(valid | visible) when <SimpleCondition>]

Detail Sections

<DetailSection>...

)

[Charts]

<ChartName> is a (BarChart | PieChart | LineChart | GapChart])

[title is <Message>]

[is default] // One chart must be designated as the default

[chart (list | elements | periods)] // default is chart list

[show fixed scale [of <RelatedValue>]]

[element name is <Message>]

[element value is <RelatedValue>...]

[title is <Message>]

[<ChartName> is a CalendarChart]

[title is <Message>]

[is default] // One chart must be designated as the default

[show date of <RelatedValue>] // Must be a Date, default is current date

[show (day | week | month)] // default is month

[legend state is <RelatedField>]

[date [range] is <RelatedValue> [to <RelatedValue>] // Date or TimeStamp

[text is <Message>]

[mouse over text is <Message>]

[raise <AlertType> alert when <SimpleCondition>...]

[mouse over text is <Message>]

[<ChartName> is a ScatterChart]
[title is <Message>]
[is default]
[point size is <RelatedValue>]
[legend state is <RelatedField>]
x-axis value is <RelatedValue>...
[title is <Message>]
[show fixed [integer] scale [of <RelatedValue>]]
[minimum value is <RelatedValue>]
y-axis value is <RelatedValue>...
[title is <Message>]
[show fixed [integer] scale [of <RelatedValue>]]
[minimum value is <RelatedValue>]

<ChartName> is a QuadrantChart
[title is <Message>]
[row title is <Message>]
[column title is <Message>]
[is default]
[show unplaced elements] // displays a list on the side of all the elements that did not have a valid quadrant address
[header is <Message>]
[implements DragAndDrop]
action is [<StateName>.,]<ActionName>
[invoked.<FullFieldName> = [drop target.]<RelatedValue>...]
rows are <RelatedLink>
[header is <Message>] // field context is target busclass of RelatedLink
[mouse over text is <Message>]
columns are <RelatedLink>
[header is <Message>] // field context is target busclass of RelatedLink
[mouse over text is <Message>]
quadrants are <RelatedLink>
[header is <Message>] // field context is target busclass of RelatedLink
[mouse over text is <Message>]
quadrant value is <RelatedValue>
row position is <RelatedValue>
column position is <RelatedValue>
[raise <AlertType> alert when <SimpleCondition>...]
[mouse over text is <Message>]

DetailSection ::= 
[<DetailSectionName>]
[mouse over text is <Message>]
[title is <Message>]
[(valid | visible) when <SimpleCondition>]
[(valid | visible) for dimension <DimensionField>]
(<Form> | <List> | link is <HttpURL> | card view is <CardViewName> | <Chart>)
[<FormDefinition>]
// valid only on inline Form
[overview navigation is <NavigationName>]
[label is <Message>]

ListDisplayField ::= 
(<FormField> | <FormButton> | <CheckControl> | <FormText>)...
[sort order is (<SetName> | primary)]
[is default [descending]]
[total]
[Subtotal By]
[<RelatedField>...]
// default is all fields in sort order
// must be field in sort order
[visible when <SimpleCondition>]
[column visible when <SimpleCondition>]

// Allows the entire column to not be displayed. SimpleCondition must be based on the incoming context
// fields of this list rather than any particular instance of this business class. If the value of SimpleCondition
// is indeterminate (if is not based strictly on valid context fields of this list invocation) then the column will not display.

[link is row[<Parens><SetName><Parens>].<ListName>]
[suppress when in dimension <DimensionField>]  // valid only in a CubeView

Card View
CardViewDefinition ::=  
   <CardViewName> is a CardView  
   [left column is (<RelatedValue> | <Icon> | representative image)]  // RelatedValue must be an image  
   [display as (portrait | photo | full) image]  // default is ‘display as portrait’
   [missing image is <ImageName>]  
   main column  
      (<FormField> | <FormText>)...  
      [visible when <SimpleCondition>]  
      [button row]
      <FormButton>...
   [right column]  
      (<FormField> | <FormText>)...  
      [visible when <SimpleCondition>]  
      [button row]
      <FormButton>...

Instance Count Chart
InstanceCountChartDefinition ::=  
   <ChartName> is an InstanceCountChart  
   [title is <Message>]  // defaults to ChartName  
   Chart Fields  
      <RelatedValue>...  
      [label is <Message>]  
      [link is <UILink>]

Forms
FormDefinition ::=  
   <FormName> is a Form  
   [is primary]  
   [is drill target]  
   [when <SimpleCondition>]  
   [propagate as a drill]  
   [use for (<DisplayType> | action [<StateName>].<ActionName>)]...  
   [title is <Message>]  // defaults to FormName  
   [Actions]  
      <CalledOutAction>...  
   [restrict action [<StateName>].<ActionName>...]  // valid only on configuration  
   [when <Condition>]  
   Layout  
      <FormLayout>...
Base Definitions

FormButton ::= button of (<Message> [with <Icon>] | <Icon>)
   [(valid | visible) when <Condition>]
   (link is <UILink>)
   | sub form is (<FormName> | primary | inline)
      (<FormDefinition>)]
   // valid only on inline Form
   | <Form> // button is implicitly invalid if in create mode and target is afforded by current instance
      [create action is [<StateName>..]<ActionName>]
      [open action is [<StateName>..]<ActionName>…]
      (<FormDefinition>)]
   // valid only on inline Form
   | <List> // button is implicitly invalid if in create mode and target is afforded by current instance
      | form invoke is <FormInvokeName>
      | action is [<RelatedLink>..]<StateName>..<ActionName>
      | show editor for <RelatedField> // Can be a BinaryDocument, GroupField, or ArrayField
      using mime type <RelatedValue>
      | show preview of <RelatedField> // Must be an Image
      using mime type <RelatedValue>
      | export <RelatedField> // Must be a BinaryDocument
      |...
      [when <SimpleCondition>] // Must have a condition if defining multiple targets
      [restrict action <StateName>..<ActionName>…] // valid only on form invoke
      [<Label>] // default is no label
      [(left align | center | right align)] // left align is default unless used in a MatrixForm (center is default)
      [(top align | middle | bottom align)] // top align is default
      [align as label] // make button line up in label column
      [indent]
      [display as link]
      [raise <AlertType> alert when <SimpleCondition>…]
      [mouse over text is <Message>]
      [hidden]

FormField ::=<RelatedField> | effective date | reason code | action comment)
   // ‘effective date’ is a Date field that will set the ‘as of’ date context for any action executed from the form this is on
   [(label is (<RelatedField> | <Message> | number) | no label)] // default is Field Label
      (<TextStyle>…]
   [(left align | center | right align)] // left align is default
   [(top align | middle | bottom align)] // top align is default
   [align as label] // make field line up in label column – valid only with no label
   [form is <FormName>] // valid only on Group and Array fields
   [indent]
   [display (only | as text)] // default to ‘only’ on SummaryForm, ListDefinition
   [hidden entry] // same behavior as Password field but no encryption
   [allow update] // default on non-related field within a normal Form
   [link is <UILink>] // valid only when display only
   [implements ManualRepresentation]
      representation name is <Literal>
      show up to <Literal> lines
      show up to <Literal> characters
      Property Mapping
      <MessageProperty> = <RelatedValue>…
      [compact format [button of <Message>]]
      [select is [<RelatedLink>..]<ListName>] // valid only on key field
      [constraint <Condition>] // valid only on configuration
      [Message]
[use text area [of <Literal> lines]] // Use a text area control to display this field
[show up to <Literal> characters] // default is 50
[keypad is (email | telephone | url)]
[prompt for (image | video | audio | document)]
[display as (hijri | gregorian) date]
[display as [(portrait | full)] image [up to <Literal> lines]]
[missing image is <ImageName>] // defaults to ‘display as portrait’; default number of lines is 8
[title is <FieldName>] // if Array of GroupFields must specify image field on GroupField and may specify title
[display as switch] // valid only a Boolean type field
[display as slider]
range is from <RelatedValue> to <RelatedValue>
[step is <RelatedValue>]
[<AlertType> alert range is from <RelatedValue> to <RelatedValue>...]
[mouse over text is <Message>] // not required when Percent type field
[meter range is <RelatedValue>] // not required when Perc
[<AlertType> alert range is from <RelatedValue> to <RelatedValue>...]
[mouse over text is <Message>]
[show as <StatusIcon> when <SimpleCondition>...]
[mouse over text is <Message>]
[display when blank] // On a SummaryForm the default is to not display blank fields
[display zero] // display ‘0’ if value is blank
[display as duration] // can be a numeric field in seconds, Time, Date, or TimeStamp
[display as byte size] // must be a numeric field
[display using <JavaFormat>] // must be a States or UserDefinedStates field
[display as [horizontal] radio buttons] // must be a TimeStamp or Time field
[<States>] // must be a TimeStamp or Time field
[hidden] // valid only on configuration
[required] // valid only on configuration
[depends on <FieldName> [,, <FieldName>]...] // FullFieldName must be on Form or List
derived value is <RelatedValue>
[when value changed]
( <RelatedField> = <RelatedValue> // RelatedField must be a mutable field on a form or list
 | refresh <RelatedField> // RelatedField must be on a form or list)
)

FormLayout ::=<LayoutDirective>

{| <FormField> | <FormButton> | <CheckControl> | <FormText> | <MultiSelectField> | <FormLayout>...}
MultiSelectField ::= multi-select field is <FullFieldName> // valid only on key field or array key field

   {{label is (<RelatedField> | <Message>) | no label}} // default is Field Label

   <TextStyle>…

   [display field is <RelatedField>]

   [select is [<RelatedLink>].<ListName>]

   [use <Literal>.<LName> lines]

   [when value changed]

   ( <RelatedField> = <RelatedValue> // RelatedField must be a mutable field on a form or list

   | refresh <RelatedField> // RelatedField must be on a form or list

   )…

Action

ActionFormDefinition ::=<FormName> is an ActionForm

   [use for <DisplayType>]

   action is [<StateException>.]<ActionName>…

   [on completion navigate to <NavigationName>]

   // An ActionForm is used to format parameters on an action. Can also be used for non-parameter-based actions.

   // When this is specified and there are parameters on the action, the field scoping is limited to the parameters.

   // If no parameters are defined on the action and the action is a Create or an Update action, the field scoping is the

   // whole business class. This kind of form only has an OK and a Cancel button. OK runs the specified Action and

   // Cancel returns to the previous page.

   // Multiple action can be defined as long as the Parameters are the same

   [allow non-modal interaction]

   // Action Forms by default are modal – they block interaction on other forms until the form is dismissed

   [title is <Message>]

   // defaults to FormName

   Layout

   <FormLayout>…

Composite

CompositeFormDefinition ::=<FormName> is a (CompositeForm | WizardForm)

   [is primary]

   [is drill target]

   [when <SimpleCondition>]}
[summary form is <FormName>]  // valid only on WizardForm—acts as a 'commit' boundary
[propagate as a drill]
[use for (DisplayType | action [StateName].)<ActionName>]]...  // defaults to FormName
[title is <Message>]  // allows common 'context' fields to be put outside the panels—not valid on Wizard
[context form is <FormName>]  // valid only on CompositeForm
[implements IndependentFormPanels]
[implements DashboardView]
[show context form on top]
[show panel control on left]
[show panel navigation]
[show steps]  // valid only on WizardForm—displays Steps Graphic
[title is <Message>]  // defaults to FormName
[icon is <Icon>]
[(valid | visible) when <Condition>]  // valid only on a Dashboard
[overview navigation is <NavigationName>]  // valid only on a Dashboard
[title is <Message>]
[mouse over text is <Message>]
[(label is <Message>)]
(CompositeFormPanel | CompositeFormMultiListPanel)...  

CompositeFormPanel :=
<PanelName> is a Panel...
[title is <Message>]
[icon is <Icon>]
[(valid | visible) when <Condition>]
[mouse over text is <Message>]
[panel header is <Message>]
[panel text is <Message>]
[overview navigation is <NavigationName>]
[title is <Message>]
[mouse over text is <Message>]
(CompositeFormPanel | CompositeFormMultiListPanel)

FormPanelDefinition ::=  
[visited action is [StateName].]<ActionName>  // valid only when 'show panel navigation' is on
[in progress when <SimpleCondition>]  // valid only when 'show panel navigation' is on
[completed when <SimpleCondition>]  // valid only when 'show panel navigation' is on
[raise AlertType alert when <SimpleCondition>....]  // valid only when 'show panel navigation' is on
[mouse over text is <Message>]
(<Form> | <List> | <Template> | <Navigation> | <URL> | <Chart>)
[<FormDefinition>]  // valid only on an inline Form
save on next  // valid on panel navigation and wizard
(<FormPanelActions>)

FormSubPanelDefinition ::=  
[visited action is [StateName].]<ActionName>  // valid only when 'show panel navigation' is on
[in progress when <SimpleCondition>]  // valid only when 'show panel navigation' is on
[completed when <SimpleCondition>]  // valid only when 'show panel navigation' is on
[raise AlertType alert when <SimpleCondition>....]  // valid only when 'show panel navigation' is on
[mouse over text is <Message>]
sub panels are <RelatedLink>
title is <Message>
<FormPanelDefinition>

CompositeFormMultiListPanel ::=  
<PanelName> is a MultiListPanel...
[title is <Message>]
[(valid | visible) when <Condition>]
[mouse over text is <Message>]
<List>...  // valid only on WizardForm
FormPanelActions ::=  

[Special Actions]  

<CalledOutAction>...  

// allows actions from the primary business class to be called from this context

Example of CompositeForm Definition

SourcingEventDocument is a CompositeForm  

title is "<mode>EventDocument"  
Event is a Panel  
form is SourcingEventHeader  
Questions is a Panel  
form is SourcingEventQuestion set  
Terms is a Panel  
list is SourcingEventTerm set  
Meetings is a Panel  
list is SourcingEventMeeting set  
Attachments is a Panel  
list is SourcingEventAttachment set  
Contacts is a Panel  
list is SourcingEventContact set  
Lines is a Panel  
list is SourcingEventLine set.EventLines  
Notifications is a Panel  
list is SourcingEventNotification set  

Matrix

MatrixFormDefinition ::=  

<MatrixForm> is a MatrixForm  

([title is <Message>]  
[display negative amounts using parens]  
[display amounts in (thousands | millions)]  
[show in title]  

[row search field is <FullFieldName>]  

[implements PeriodFilter]  
	date is <FullFieldName>  
	(monthly | weekly | daily) period is <FullFieldName>  
	year is <FullFieldName>  

[show (day | week | month | quarter | year)]  
[current (date | period) is <RelatedValue>]  

current year is <RelatedValue>  

date is <FullFieldName>  

context form is (<FormName> | inline)  

[<FormDefinition>]  

// valid only on inline Form

Column

Display Fields  

(<FormField>...  
| <FormButton>...
Cell
[overview navigation is <NavigationName>]
[label is <Message>]
Display Fields
{ <FormField>...
  [(valid | visible) when <SimpleCondition>]
  | <FormButton>...
  | <CheckControl>...
}  
Detail Fields
{ <FormField>...
  | <FormButton>...
  | <CheckControl>...
}

Example of Matrix Definition

CompareResponses Is a MatrixForm
title is “CompareResponses”
context form is CompareResponsesContextForm

Column
  Display Fields
  Supplier.SupplierName
    no label
    check control
      check action is AwardAllLines
      uncheck action is UnAwardAllLines
  SupplierTotalBidAmount
  link is QuestionResponses
    label is “ViewResponses”

Cell
  Display Fields
    check control
      label is “Line<SourcingEventLine>:<SourcingEventLine.Description>”
        header:2
        checked state is IsAwarded
        check action is AwardLine
        uncheck action is UnAwardLine
    BidAmount
      label is “<SourcingEventLine.LineInfo>”
  Detail Fields
    ExtendedPrice
    RequestedDeliveryDate
    link is LineQuestionResponses
      label is “ViewResponses”
Search

SearchFormDefinition ::= 
  <FormName> is a SearchForm 
  [title is <Message>] // defaults to FormName 
  Layout 
  <FormLayout>...

Summary

SummaryFormDefinition ::= 
  <FormName> is a SummaryForm 
  [title is <Message>] // defaults to FormName 
  [disallow show as pdf] 
  [First Page Header] 
    ( form is <FormName> 
      | Layout 
      <SummaryLayout>...
    )
  [Page Header] 
    [print on first page] // not valid if First Page Header is defined 
    ( form is <FormName> 
      | Layout 
      <SummaryLayout>...
    )
  [Page Footer] 
    ( form is <FormName> 
      | Layout 
      <SummaryLayout>...
    )
  Layout 
  <SummaryLayout>...

SummaryLayout ::= 
  ( <FormField> 
    | <FormButton> 
    | <CheckControl> 
    | <FormText> 
      | form is [RelatedLink].<FormName> // can take only an OTM RelatedLink 
      | [Header] 
      | list is [RelatedLink].<ListName> // can take only an OTM RelatedLink 
        [show grid lines] 
        | [Header] 
        | <LayoutDirective> 
          [SummaryLayout>...]
    )

Page Definition

// Page definition files must have an extension of .page and are found in com/lawson/forms/<ModuleName>. Page names exist in the same name space as // BusinessClass names and BusinessTask names.
/ Pages and CompositeForms have a lot of similarity but also some significant differences.
// The similarity is that they are both made up of panels.
// The difference is in their ontology and their (current) capability.

// A CompositeForm is defined within the scope of a BusinessClass, BusinessTask, or Field, like a regular Form. Because of this it presupposes that all references
// are references within the context of that Business Class. A Page is defined as an individual entity at the same scoping level as a BusinessClass, Task, or Field.
// It is also made up of panels but each panel must designate its own particular context. Pages can also have multi-pane panels whereas CompositeForms cannot.
// This restriction is not ontologically required; it is just a simplification for now.

Page Structure ::= 

<PageName> is a Page
    [title is <Message>] // defaults to PageName
    [allow anonymous access]
    [show panel control on left]
    [Context Form]
    Layout
        <FormLayout>... // valid fields are the set of all Context fields defined on all Panels and panes

<PanelName> is a Panel ...
    [title is <Message>] [mouse over text is <Message>]
    <PanelDefinition>

<PanelName> is a DashBoardPanel ...
    [title is <Message>] [mouse over text is <Message>]
    <PaneName> is a Pane...
    <PanelDefinition>

<PanelName> is a MultiPanePanel ...
    [title is <Message>] [mouse over text is <Message>]
    [vertical split is <Number>/<Number>] // default is 50/50
    pane <PaneNumber>...
    <PanelDefinition>

<PanelName> is a ColumnarPanel ...
    [title is <Message>] [mouse over text is <Message>]
    [left column]
        <PanelDefinition>... // can have multiple panels if each one is a FixedSizeList
    main column
        <PanelDefinition>...
    [right column]
        <PanelDefinition>...

<PanelName> is a WorkViewPanel
    [title is <Message>]
    pane <WorkViewPaneNumber>...
        (<WorkViewPanel> | <MenuPanel>)

PaneNumber ::= (1 | 2 | 3 | 4) // 1 is upper left, 2 is lower left, 3 is upper right, 4 is lower right

// The current metaphor for MultiPanePanel panes is that there are four possible panes: pane 1 is the upper left, pane 2 is the lower left,
// pane 3 is the upper right and pane 4 is the lower right.
// Pane interaction is accomplished through inferencing how the actions relate to each other through the lpl business class action specification.
PanelDefinition ::= ( <ListPanel> | <MenuPanel> | <SearchPanel> | <URLPanel> )

PanelDefinition ::= ( <ListPanel> | <MenuPanel> | <SearchPanel> | <URLPanel> )

[(valid | visible) when <SimpleCondition>]
// SimpleCondition must start with SessionKey or actor.context.KeyField
// This is not valid when the Panel is used in a Multi-Pane Panel

ListPanel ::= business class is ( <BusinessClass>[(<AsOfOperator> | .<RelatedLink>)]
| pane<PaneNumber>..<RelatedLink>
| <Agent>..<RelatedLink>
)

ListPanel ::= business class is ( <BusinessClass>[(<AsOfOperator> | .<RelatedLink>)]
| pane<PaneNumber>..<RelatedLink>
| <Agent>..<RelatedLink>
)

[(list is (<ListName> | inline))] // defaults to primary list on resultant business class
<title is <Message>] // defaults to the title of <ListName>
[display (list | <ChartName>)] // overrides the list default
[Row Dimensions] // valid only on a CubeView, overrides the CubeView Row Dimension
<DimensionField>...
[overview navigation is <NavigationName>] // valid only on a DashBoardPanel
[context is <Message>] // valid only on inline List

[helper list is <RelatedLink>.(<ListName> | primary)] // valid only on a relative pane - <RelatedLink> is relative to pane<PaneNumber>
[
valid when <SimpleCondition>]
[actions is [<StateName>..<ActionName>...]]
[invoked.<<FullFieldName> = <RelatedValue>]

[Context]
<FieldName>... // Must be an ontological context field of the business class or, if the list is a CubeView, it can
// also be a dimension. Values for this can come from a context session key, an
// 'actor.agent(BusClass).FieldName' lookup, or from the current context before a
// Navigation brings up this page

[Actions]
( <CalledOutAction>
| form is <RelatedLink>..<FormName> // <RelatedLink> can only be one of the context fields
  | <Label>
)...

MenuPanel ::= Layout
[footer is <CardViewName>] // must be first element under Layout
<MenuLayout>...

MenuLayout ::= ( <FormText> | <MenuItem>

<MenuItem> ::= ( page is <PageName>.[.<PanelName>] | list is <BusinessClass>[(<AsOfOperator>].(<ListName> | primary)

40 | Configuration Console Reference Guide
| form is `<BusinessClass>.(<FormName> | primary)`
  [actor agent required]
  [create action is `[<StateName>].<ActionName>]
  [open action is `[<StateName>].<ActionName>`]
  [restrict action `[<StateName>].<ActionName>`]
| action is `[<BusinessClass> | <BusinessTask>].<ActionName>`
| link is `<BusinessClass>.[<UILink>`   // except NavigationName when no BusinessClass
| menu is `<MenuName>`
| image map is `<ImageMapName>`
| webapp is `<WebAppName>`   // valid internally for configuration only
)

// `form is `<BusinessClass>`...`, `list is `<BusinessClass>`...`, `link is `<BusinessClass>`...` and `action is `<BusinessClass>`...` will try to find the most relevant instance of
// the specified `<BusinessClass>` in their context. They first look for an agent of `<BusinessClass>` type that is linked with the current actor. Failing that they will both fill
// in their primary keys with actor context variables and session.key variables and then try to find an instance based on those keys.

SearchPanel ::= 
  search form is `<BusinessClass>..<FormName>`
  [list is (<ListName> | inline)]   // defaults to primary list on resultant business class
  [ListDefinition]   // valid only on inline List
  [Context]
  <FieldName>...
  [Actions]
  ( [action is] `[<StateName>].<ActionName>`...
    [Label]
  | form is `<RelatedLink>..<FormName>`...   // `<RelatedLink>` can only be one of the context fields
    [Label]
  )

URLPanel ::= 
  url is [(<BusinessClass> | pane <PaneNumber>).]<HttpURL>

WorkViewPanel ::= 
  ( business class is ( `<BusinessClass>`[.<RelatedLink>] 
    | pane<PaneNumber>[.<RelatedLink>]   
    | <Agent>[.<RelatedLink>] 
  )
  [title is `<Message>`]   // defaults to the title of `<ListName>`
  [show <> 1lines]   // defaults to 15 lines
  [list is (<ListName> | inline)]   // defaults to primary list on resultant business class
  [ListDefinition]   // valid on inline List only
  [sort order is (<SetName> | primary])
  [Context]
  <FieldName>...
  [Instance Selection]
  where `<Condition`
  | detail fields view of pane<PaneNumber>
  | link display view of pane<PaneNumber>
  )

WorkViewPaneNumber ::= (1 | 2 | 3)   // WorkViewPanes are columnar. There can be only up to 3 WorkViewPanes.

// A context field will implicitly be used as a ‘fixed’ context to any Create action that is executed – if Context of ‘Actor’ is specified
// and a Create action is executed, then the Actor field in the business class will be populated with the Actor context
Security Class Definition

Security Class Structure ::= 

<SecurityClassName> is a SecurityClass 
[extends <SecurityClassName>] 
[description is <Message>] 
Access Rights 
   <PolicyDefinition>...
Structure Definitions

PolicyDefinition ::= ( <AccessPolicy> | <OntologicalPolicy> )

AccessPolicy ::=<SecurableObjectSpecification> <AccessRule>...

OntologicalPolicy ::=<KeyField> KeyField <GrantRule>...

Base Definitions

ActionKeyword ::= ( all actions | all audit views | all creates | all deletes | all inquiries | all functions | all updates | all global UI configuration | all security configuration | all personalization | limited list personalization | limited form personalization | all scheduled actions | all actor groups | data menu | future data indicator | UpdateEffectiveDatedData | ViewAuditLog | ViewFullAuditLog )

ActionList ::= ( <SecurableAction> | <ActionKeyword> )

AccessRule ::= (is accessible [and attachable] | is (not | neither) accessible [nor attachable]) <SecurableActionSpecification> <Constraint>

Constraint ::= ( unconditionally | when <Condition> )

ContentSet ::= All Fields for <LMObject> [excluding <FieldName [Field]>... ] // separate list of fields with a comma if on the same line

GrantedObjectList ::= ( <SecurableObjectName> <SecurableObjectType> )
Security Class Definition

| all business classes
| all business tasks |

GrantedObjectSpecification ::= ( to <GrantedObjectList>...
| all ontology
   [excluding <GrantedObjectList>...]  // separate list of objects with a comma if on the same line
)

GrantRule ::= grants access [and attachability]
   <GrantedObjectSpecification>
   <SecurableActionSpecification>
   <Constraint>

LMContainerType ::= ( DataArea
   | Module
   )

LMContentType ::= ( Field
   | KeyField
   )

LObject ::= <SecurableObjectName> <LMObjectType>

LMObjectType ::= ( BusinessClass
   | BusinessTask
   | Menu
   | MenuItem
   | WebApp
   )

MenuItemSpecification ::= <MenuItemName> MenuItem for <MenuName> Menu

SecurableAction ::= [<SecurableObjectName>.][<StateName>.]<ActionName>

SecurableActionSpecification ::= for <ActionList>...
   [excluding <ActionList>...]  // separate list of actions with a comma if on the same line

SecurableObjectName ::= <Literal>

SecurableObjectSpecification ::= ( <SecurableObjectName> <SecurableObjectType>
   | <SecurableObjectType> Type
   | <ContentSet>
   | <MenuItemSpecification>
   )

SecurableObjectType ::= ( <LMObjectType>
   | <LMContainerType>
   | <LMContentType>
   )
Security Examples

Simplest way is security on Menu type that allows access to all Menus for all actions

Menu Type

  is accessible
  for all actions
  unconditionally

Policies on Menu and Menu Items

EmployeeSelfService Menu

  is accessible
  for all actions
  unconditionally

Directory MenuItem for EmployeeSelfService Menu

  is accessible
  for all actions
  unconditionally

MyProfile MenuItem for EmployeeSelfService Menu

  is accessible
  for all actions
  unconditionally

MyCompensation MenuItem for EmployeeSelfService Menu

  is accessible
  for all actions
  unconditionally