Configuration Console Reference Guide

Landmark Pattern Language (LPL)
May 2022

Contents

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.

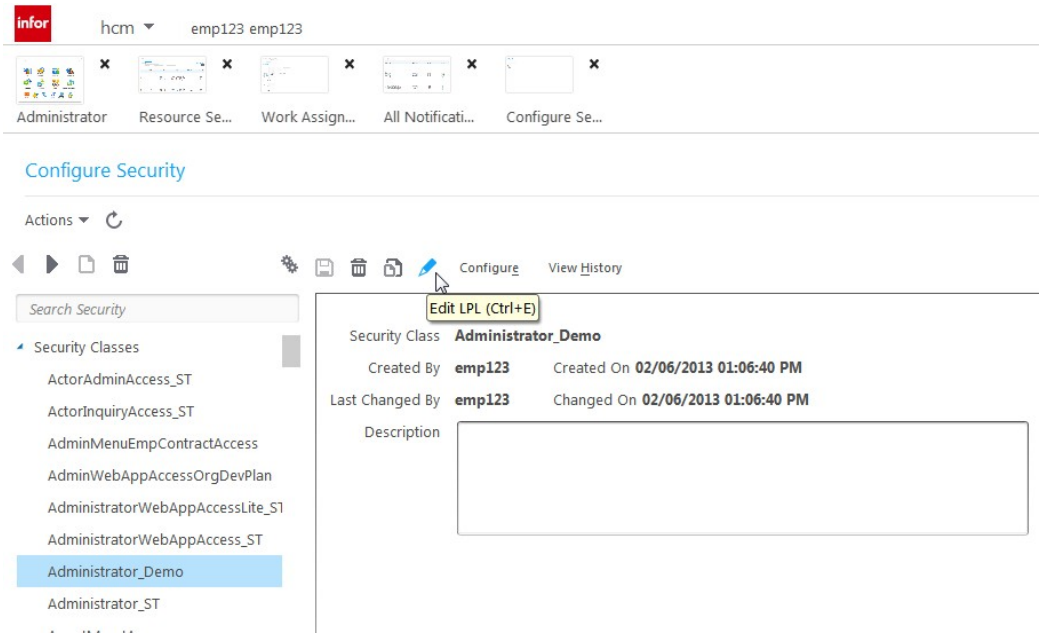
Configuration Console Development Process and the Landmark Pattern Language

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
// Comments are denoted by a double forward slash
// @SuppressWarnings in a comment will suppress all warnings that the LPL line generates
// @External with a comment will suppress dead code detection/reporting

Base Definitions

LPLConstructName ::= *<uppercase character>[<alphanumeric characters>…]* *// up to 255 characters*

ActionName ::= **<LPLConstructName>**
ActionTag ::= **<LPLConstructName>**
BusinessClass ::= **<LPLConstructName>**
BusinessTask ::= **<LPLConstructName>**
CardViewName ::= **<LPLConstructName>**
ConditionName ::= **<LPLConstructName>**
ChartName ::= **<LPLConstructName>**
ContextMessage ::= **<LPLConstructName>**
CriteriaName ::= **<LPLConstructName>**
CubeLinkName ::= **<LPLConstructName>**
CubeRelationName ::= **<LPLConstructName>**
DetailSectionName ::= **<LPLConstructName>**
DrillName ::= **<LPLConstructName>**
DrillListName ::= **<LPLConstructName>**
FieldGroupName ::= **<LPLConstructName>**
FieldName ::= **<LPLConstructName>**
FormInvokeName ::= **<LPLConstructName>**
FormName ::= **<LPLConstructName>**
FullStateName ::= **<StateName>[.<StateName>…]**
GroupName ::= **<LPLConstructName>**
HierarchyName ::= **<LPLConstructName>**
ImageMapName ::= **<LPLConstructName>**
ListName ::= **<LPLConstructName>**
MajorSystemProcess ::= **<LPLConstructName>**
MatrixForm ::= **<LPLConstructName>**
MenuItemName ::= **<LPLConstructName>**
MenuName ::= **<LPLConstructName>**
M3Interface ::= **<LPLConstructName>**
NavigationName ::= **<LPLConstructName>**
PageName ::= **<LPLConstructName>**
PanelName ::= **<LPLConstructName>**
PaneName ::= **<LPLConstructName>**
ParameterName ::= **<LPLConstructName>**
PeriodViewName ::= **<LPLConstructName>**
PFlowServiceName ::= **<LPLConstructName>**
ProgramName ::= **<LPLConstructName>**
RelationName ::= **<LPLConstructName>**
ReportName ::= **<LPLConstructName>**
RuleBlockName ::= **<LPLConstructName>**
RuleBlocks ::= **<LPLConstructName>**

```
SectionName          ::= <LPLConstructName>
SecurityClaim        ::= <LPLConstructName>
SecurityClassName    ::= <LPLConstructName>
SetName              ::= <LPLConstructName>
StateName            ::= (<LPLConstructName> | <#>)
StateCycleName       ::= <LPLConstructName>
StateFieldName       ::= <LPLConstructName>
StaticJavaWS         ::= <LPLConstructName>
StaticJavaPD         ::= <LPLConstructName>
Subject              ::= <LPLConstructName>
TableName            ::= <LPLConstructName>
TextVariable         ::= <LPLConstructName>
WidgetListName       ::= <LPLConstructName>
WebAppName           ::= <LPLConstructName>
WebserviceInterface::= <LPLConstructName>

Parens ::= ( '('
           | ')'
           )

ModuleName           ::= <alphanumeric characters>
ModuleDescription    ::= <Literal>

ClassicPrefix         ::= prefix is <string of 2 to 5 alphanumeric characters>
ClassicName           ::= classic name is <Literal>              // literal must be all uppercase, no spaces
ClassicNameForField   ::= classic name for <FullFieldName> is <Literal>

RpgName          ::= rpg name is <Literal>

SqlPrefix        ::= sql prefix is <string of 2 to 5 alphanumeric characters>     // no spaces
SqlName          ::= sql name is <Literal>                          // no leading or trailing spaces allowed
SqlNameForField ::= sql name for <FullFieldName> is <Literal>
```

Data and Field Types

```
DimensionField       ::= <dimension RelatedField>     // this includes isAggregatable attributes
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>%
ReportText           ::= (<alphanumeric characters> | ':')
Text                 ::= '"'<alphanumeric characters>'"'
ViewField            ::= <view FullFieldName>

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>]
                         [measures only]
                   | is <TypeDataDefinition>
```

```
                    )       // needs to have a standard set of 'system' fields: product line, module, and so on

PrimitiveType ::=   ( Alpha
                    | AlphaRight
                    | AlphaUpper
                    | Anniversary                                       // cannot translate
                    | BinaryDocument
                    | BinaryObject
                    | Boolean                                           // cannot translate
                    | BusinessObjectReference [to <BusinessClass>]      // cannot translate
                    | CSVText                                           // cannot translate
                    | Date                                              // cannot translate
                    | [Unsigned] Decimal                                // cannot translate
                    | DocumentTitle                                     // Alpha field with special characteristic: when
                          // in the context of a BinaryDocument, it is populated when the BinaryDocument is populated. When user selects a file on
                          // disk into a BinaryDocument field, we search for the closest DocumentTitle and MimeType and fill them out.
                    | EmailAddressField [with multiple addresses]
                          // Deprecated: "with multiple addresses" due to size limitation – use MultiEmailAddressField
                    | MultiEmailAddressField
                    | GroupField                                        // cannot translate
                    | Integer                                           // cannot translate
                    | Iteration of <BusinessClass>                      // cannot translate
                    | JSONObject                                        // cannot translate
                    | MimeType                                          // cannot translate
                    | Numeric                                           // cannot translate
                    | Password                                          // cannot translate
                    | [Unsigned] Percent                                // cannot translate
                    | Period                                            // cannot translate
                    | RichText
                    | Signed                                            // valid only in a Report Definition
                    | Text
                    | TextDocument                                      // cannot translate
                    | Time                                              // cannot translate
                    | TimeStamp                                         // cannot translate
                    | UniqueID                                          // cannot translate
                    | URI
                    | URL
                    | XMLDocument                                       // cannot translate
                    | Year                                              // cannot translate
                    )

TypeDataDefinition ::=
    [us-ascii]<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

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

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

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

CurrentActionBackgroundGroupId ::= current action background group id  // valid only when in a background action

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.

DocumentLocation ::= document location.(Local | AWSS3 | External)
                                        // State values for 'document location' TypeOperator

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

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

JavaFormat ::= '"'(<Literal> | '<'<RelatedValue>'>')…'"'
                        // Must resolve to a JavaFormat as defined here:
                        //  http://docs.oracle.com/javase/7/docs/api/java/util/Formatter.html
                        //  can also see https://en.wikipedia.org/wiki/Regular_expression

RelevanceScore::= relevance score      // this 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>
           | <StateFieldName>
           )
```

Operators

```
AsOfDate ::=     ( as of (<RelatedValue> | all dates)
                 | after <RelatedValue>
```

```
                  | before <RelatedValue>
                  | between <RelatedValue> and <RelatedValue>
                  )                          // 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
                   | any                              // valid only with multi-valued (array, otm) RelatedField or RelatedCondition
                   | all                              // "
                   | first                            // "
                   | last                             // "
                   | sum                              // "
                   | avg                              // "
                   | min                              // "
                   | max                              // "
                   | floor
                   | ceiling
                   | pending
                   | instance count of                // "
                   | sizeofarray
                   | (first | last) iteration of      // valid with a OTM RelatedLink only
                   | next                             // valid on Iteration fields only
                   | previous                         // "
                   | reference to                     // valid only with a OTO RelatedLink only – returns the BusinessObjectReference
                                                      //     of the RelatedLink
                   | target of                        // valid only with an "is related value…" designated field
                   )

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 in a CubeView

TotalOperator ::= ( running total
                  | percent of total)

TypeOperator ::=  ( decimals                  // valid with a Decimal type field only
                  | type                      // field type – returns string of <PrimitiveType; for example, Alpha, Decimal, Numeric…
                  | size                      // size of  the 'string' value of a field
```

```
| uppercase
| lowercase
| 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              // "
| week day                      // " – 1-based week day number (1-7)
| corporate week day                  // week day based on corporate TimeZone
| week day using <RelatedValue>       // week day based on RelatedValue TimeZone (must be a TimeZone)
| 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)
| corporate date                // valid with TimeStamp type fields only (returns Date in corporate time zone)
| system date                   // valid with TimeStamp type fields only (returns Date in system's default time zone)
| date using <RelatedValue>// valid with TimeStamp type fields only, RelatedValue must be a TimeZone
| 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
| level <#>                    // valid with key field that is a hierarchy; specifies which level in the hierarchy to address
| 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
| [full] name                  // returns the LPL name or full name of the field
| text                         // valid only with a field that has Text Variable; retrieves the text with the variables replaced
| [exact] translation          // returns the data translation for the current local – "exact" will prevent standard defaulting
| mime type                    // valid with BinaryDocument and BinaryObject only
| in base64                    // valid with BinaryDocument and BinaryObject only
| translations                 // returns the set of data translation for a field; valid on right side of assignment operator only
| document location            // returns <document location.(Local, S3, External)>; can be modified
| (local | external) document      // returns the local or external document; external includes AWSS3 location
| select <SelectStatement>
| select line <SelectLineStatement>
| 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'
                                // if RelatedLink is an action request id, document will be replaced based on the action request
| compute value [for <RelatedLink>]
                                // valid with a field that is a BusinessClassCompute only
| 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>
| cube dimension value [using year of <RelatedValue>]
                                // a dimension value in a cube is typically different than the BusinessClass value
```

```
                                    // a Period dimension that is not satisfied with a date must have a year specified
| input value                       // always returns the input value on a transient field with a derived value
| plain text                        // strips all formatting (except new line) from RichText fields
| split [on <Literal>]              // splits string into an array of strings using 'Literal' as separator (comma is default)
| using <JavaFormat>
| (short | medium | long) timestamp [with day [short] name]
                                                  [and][timezone][hide seconds]
                                    // valid with TimeStamp – returns a String
                                    // http://docs.oracle.com/javase/7/docs/api/java/text/DateFormat.html
                                    // https://docs.oracle.com/javase/tutorial/i18n/format/dateFormat.html
| (short | medium | long) date [with day [short] name]
                                    // valid with TimeStamp and Date – returns a String
| (short | medium | long) anniversary [with day [short] name]
                                    // valid with TimeStamp, Date, and Anniversary – returns a String
| (short | medium | long) period
                                    // valid with TimeStamp, Date, and Period – returns a String
)
```

Dates and Time Stamps

```
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 [(corporate | user)] time
                     | [system] current [(corporate | user)] date
                     | [system] current [(corporate | user)] year
                     | [system] current [corporate] period
                     | [system] current timestamp
                     | [system] current [(corporate | user)] 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
```

```
ConfigCategory ::= <Literal>

ConfigVar ::= [stack]config[<Parens><ConfigCategory><Parens>].<Literal>
                        // Allows reference to a configuration variable. Configuration variables are defined in the BusinessClass
                        // ConfigurationParameter - if it is not found it is blank
                        // 'stackconfig' checks the parameter at the dataarea, then the tenant, then the stack

DataLink ::= [<LinkOperator>]
             ( <KeyField> [set]
             | <hierarchy KeyField> ( parent
                                    | [and] children
                                    | [and] siblings
                                    | [and] descendants
                                    | [and] ancestors
                                    | ascendant
                                    )
             | <RelationName>
             | <CubeRelationName>
             | <BusinessClass>[<Parens>(<KeyField> | <BusinessObjectReference>)<Parens>] [set]
             | <BusinessObjectReference>[<Parens><BusinessClass><Parens>]
             | <BusinessClassTotal>
             | <Agent>
             | <SessionKey>
             | actor.context.<KeyField>
             | audit log records [descending]                          // retrieves the set of past and future (effective-dated) records
             | [(draft | in process | completed | rejected)] [<BusinessClass>.Create] action requests
             | this instance
             | related                                                 // valid within a Relation Definition
             | from                                                    // valid within a where condition on an OTM DataLink
             | each[<Parens><LPLConstructName><Parens>]                // valid within a for each loop
             | invoked                                                 // valid within an invocation
             | result                                                  // valid within an invocation
             | child                                                   // valid within Parent Attach Rules
             | cube[<Parens><BusinessClass><Parens>]                   // used to address the AnalyticCube business class
             | cube drill set                                          // used to address the drill set from an AnalyticCube
             )
             [<AsOfOperator>]
             [<Parens>locale of <RelatedValue><Parens>]                // RelatedValue must be an IsoLocale
             [<Parens>where <Condition><Parens>]
             [<Parens>level <#><Parens>]                               // only valid on a hierarchy KeyField
                        // <RelatedLink> option is to indicate that you can string related links, that it is recursive;
                        //     for example, KeyField.RelationName.KeyField.RelationName.Field
                        // The BusinessClass in a 'BusinessClass set' must either have a 'part of' ontology to
                        //     the preceding item or it must have a field that is one of the symbolic keys of the preceding item
                        // If there is more than one field that matches the preceding items symbolic keys then
                        //     (<KeyField>) must be present to disambiguate which one is desired
                        // A BusinessObjectReference must have a BusinessClass designation if it has not been defined
                        //     as a particular kind of BusinessClass
                        // If this business class is an Agent then another agent that has been linked to this agent can be
                        //     addressed with the 'agent(<BusinessClass>)' designation. To access any agents that the current
                        //     actor has been linked with simply add the 'actor' keyword in front of the agent keyword.
                        // 'cube' must be the first item in any RelatedLink. It refers to this business class' AnalyticCube unless this is
                        //     overridden with a specific BusinessClass inside the parens.

Distinct ::= distinct (<FullFieldName> | <FieldGroupName>)
                                    // Currently used in 'for each' rule only. Returns a set of instances where the Field (or Fields)
                                    // are distinct–that is, no duplicates of Field (or Fields) will be returned. The instances
                                    // returned will have only the distinct fields filled out as well as any fixed fields, such as higher
                                    // level keys that are fixed in the related definition.

DocTemplate ::= (template.<Literal> | template name.<RelatedField>)
```

```
                                    // Allows reference to an XML, JSON, Word rtf, or PDF template stored in the UserTemplate business class

InlineEquation ::= [<Parens>]<RelatedValue> <MathOperator> <RelatedValue>[<Parens>]
                                    // A basic InlineEquation (A + B) can also be considered a RelatedValue, because InlineEquation is an option of
                                    // RelatedField, which is an option of RelatedValue. This means it can be used as a RelatedValue in the more complex
                                    // InlineEquation ((A + B) + C), where (A + B) is the first <RelatedValue> in the InlineEquation, the second + is
                                    // the <MathOperator>, and C is the second <RelatedValue>.

LogicalID ::= logical id  // logical id is set in LogicalIdMapping

ParentContext ::= parentcontext.( isbusclass
                                | name
                                | locale
                                | stereotype
                                | istransaction
                                | module
                                | dataarea
                                | <SymbolicKeyVar>
                                | isbustask>
                                )
                                    // 'parentcontext' refers to the parent context of the current context
                                    // symbolickey retrieves the context information of the Symbolic Key that this business object is 'centered' on
                                    // If the business object is not defined directly via a Symbolic Key but is defined as being 'part of' some other
                                    // business class then it gets the context information of the Symbolic Key of that business class.

Phrase ::= phrase[<Parens>locale of <RelatedValue><Parens>].<Literal>
                                    // Allows reference to a translatable phrase. Phrases are defined in the UserPhrase business class.

RelatedCondition ::= [<FieldOperator>]( [<RelatedLink>.]<ConditionName>
                                      | <RelatedField>.<StateName>
                                      )

RelatedField ::= [<FieldOperator>] +
                 (
                 [<RelatedLink>.]<FullFieldName>
                                    // <FullFieldName> is optional with the FieldOperators instance count of and (first | last) iteration of
                 | <RelatedLink> (first | last) date within <RelatedLink>
                                    // Second RelatedLink must point to a BusClass group. The result of this statement is a date (never a
                                    // condition) so 'first' or 'last' is required. For example, 'Employee date within EnrollmentGroup'
                                    // implies a condition whereas 'first date within' clearly implies that the result is a date.
                 | instance count [where <Condition>]
                                    // valid in List Definition Summary Total Fields and ColumnarView lists only
                 | <InlineEquation>
                 ) [<TypeOperator>] [<TotalOperator>]

RelatedLink ::=   <DataLink>[.<DataLink>…]

RelatedValue ::= ( <RelatedField>[.<StateName>]     // StateName is one of the States defined on the RelatedField
                 | <Value>
                 | <ParentContext>
                 | <Actor>
                 | <CurrentDateTime>
                 | <DurationVars>
                 | <CreateStamp>
                 | <UpdateStamp>
                 | <ConfigVar>
                 | <DocTemplate>
                 | <Phrase>
                 | <URLVar>
                 | <SessionKey>
```

```
                    | <SessionClaim>
                    | <LinkBack>
                    | <BaseURL>
                    | error message [(key | field name | stack trace)] // valid with InvokeRule 'resume on error' only
                    | page number                                      // valid in Page Header or Page Footer only
                    | <NavigationName> as pdf [in (portrait | landscape)] [font offset is <Number>]
                          // default is portraitC
                    | <PeriodLabel>
                    | <CreateOrUpdateMode>
                    | <TenantID>
                    | <LogicalID>
                    | <DocumentLocation>
                    | <TotalVar>
                    )

SessionClaim ::= session.claim.<Literal>
                          // Allows reference to a session claim. Session claims are typically Security Claims set on the session context.

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')

SelectStatement ::= <Message>
                          // if the field is an XMLDocument, the SelectStatement is interpreted as XPath
                          // if the field is a JSONObject, the SelectStatement is interpreted as JSON
                          // if the field is any other type, the SelectStatement is treated as a Regular Expression

SelectLineStatement ::= that (starts with | contains | ends with) <Message>

TenantID ::= tenant id

TotalVar ::= ( total text                 // references "sub-total text is" and "grand total text is" from ListDefinition – used on print total form
             | total key[.representative text])
                                          // can be used in "sub-total text is" and "grand total text is" <Message> in ListDefinition

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
```

Discussion of Context

In general a context field is similar to a parameter on a function. The key difference is that a parameter on a function must be explicitly passed and is therefore 'hard-wired' whereas a context field automatically searches its 'context' to find the appropriate field to hook up to. If the Employee field has a context field of Company, then when it is put on some business class, it searches the fields on that business class seeking to find one that matches up with Company. A match is found when the type of the context field matches the type of some target field. The context field will match if it finds a supertype but will not match if it finds a subtype. Therefore APCompany matches to Company but Company does not match to APCompany. The search must find an unambiguous reference within a field group level. The search starts with the local field group level and if a match is not found it goes up a level. It keeps doing this until it either finds a match or runs out of levels. The search never goes down a level or into a GroupField at the same level, which is considered to be going down a level.

There are two key usages of context fields: ontological and regular. Ontological context fields define the affordance structure of a KeyField. These fields are only resolved within the context of a business class. Non-optional ontological context fields must be resolved at compile time or an error will result. If a regular context field does not find a match within the context of its business class then it will match dynamically based on its usage. It will dynamically go outside of the context of its business class to find a match. For example, WorkersCompClass is a KeyField with a regular context field called EffectiveDate. It is placed on the business class JobCode. Because Jobcode does not have an EffectiveDate on it the context field is dynamically resolved. The business class TimeRecord has a JobCode KeyField on it, and an EffectiveDate. The reference JobCode.WorkersCompClass.Rate from the context of the Timerecord business class causes the WorkersCompClass field to search TimeRecord's fields at the same level and above as JobCode and hooks up with TimeRecord.EffectiveDate.

A context field can have rules associated with it to determine its value. The special field operator in context is provided to allow for rules based on whether the context field is found in the context or not.

Example

```
PeriodCalendar is a Field

    Representation
        Group Fields
            FiscalYear
            NbrPeriods
            PeriodEndDates

    Context
        RunPeriod is an AccountPeriod
            if (RunPeriod not in context)
                value is index of first PeriodEndDates.Date >= RunDate

        RunDate is a Date
            if (RunDate not in context)
                if (RunPeriod not in context)
                    value is current date
                else
                    value is PeriodEndDates.Date[RunPeriod]
```

Actions and Actors

```
ActionAttribute ::= ( <Actor>          // action attributes either reference the currently executed action or an audited action
                    | 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
                        | action comment
                        | action type[.(Create | Update | Delete | Unknown)]
                                                          // value is true if the action type is one of the listed types
                        | action tag                      // designer defined 'tag'
                        | applied stamp                   // logical timestamp of when record actually changed / action was 'applied'
                        | audit entry id                  // if active entry – the id of the entry
                        | audit period[.(Past | Current | Future)]
                        | correction                      // true if active entry is a correction
                        | correction comment              // if correction – the comment
                        | effective date
                        | effective time zone             // time zone for effective date – default is server time zone
                        | effective through               // if new entry correction – the effective through date
                        | effective stamp
                        | entry stamp                     // logical timestamp of change 'request'
                        | initiating action
                        | invoking action                 // action immediately preceeding this one in the invoke chain
                        | reason code
                        | subject
                        | system stamp                    // physical timestamp of action
                        | action request id
                        | changed field names             // comma separated list of changed field names
                        | changed fields                  // list of changed fields – valid only in 'for each' rule
                        | purge date                      // date all data was purged before
                        | audit transaction id            // id that ties all audit log entries done in a transaction
                        | session                         // currently not in use
                        | server identity                 // set of IP addresses for the server that ran the action
                        | remote identity                 // IP address for the remote client that ran the 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>
                                           | context.<SecurityClaim>
                                           )]
                                // '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)
                                // if this is used when doing a 'for each audit log records' then it refers to the actor of the particular audit log action

ActorAttribute ::=    ( <actor FieldName>                 // Field on Actor.busclass in environment product line
                        | initiator                       // returns true if the actor is the intiator of a RequestAction
                        | approver                        // returns true if the actor is an approver of a RequestAction
                        | final approver                  // returns true if the actor is a 'final approver' of a RequestAction
                        )

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

LinkBack ::= linkback<Parens>                             // The linkback statement is a single line statement. These options have been
                        webapp is <WebAppName>            // placed on separate lines in the BNF for readability.
                        navigation is <NavigationName>
                        [allow anonymous access]
```

```
                    [show form only]          // Suppresses display of the header and navigation bar
                    [text is <Message>]
                    [session key <KeyField> is <RelatedValue>…]
                <Parens>                      // 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.
```

Messages

```
IsoLocale ::= <Literal>    // must be a valid IsoLocale

LiteralMessage ::= '"'( <Literal> | '<'<RelatedValue>'>')…'"'
                           // 'CompanyIsRequired' will display the message 'Company is required'
                           // 'Company<Company>IsRequired' will display the message 'Company 123 is required'

MessageID ::= '"'( <Literal> | '<'<RelatedValue>'>')…'"'
                           // 'CompanyIsRequired' will display the message 'Company is required'
                           // 'Company<Company>IsRequired' will display the message 'Company 123 is required'

Message ::= ( [(untranslatable: | configuration:)]<LiteralMessage>]
            | <MessageID>
            )
                [<MessageTranslation>…]          // valid only when 'configuration:' used
                                                 // 'untranslatable' is valid in all LPL
                                                 // 'configuration' valid in a configuration only

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

Arrays

```
ArrayComponent ::=   ( <array RelatedField>  // This can also be a simple Alpha field which then implicitly addresses each character
                     | <array RelationName>
                     | <ArrayRangeAddress>
                     | <special index SingleValueArrayAddress>
                     )     // A Field or a Relation can be an array. A Relation becomes an Array Relation when one of the Values
                           // 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
                           // dimension array or a two dimension array.

ArrayRangeAddress ::= <ArrayComponent>'['[<I>=]<from IndexVariable>:<to IndexVariable>']'
                           // 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>'['<IndexVariable>']'
// 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).

Examples of Array Expressions

```
A = ArrayField[5]
```

```
A = ArrayRelation[1].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[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.

Field Definition

// Field files have an extension of .field or .keyfield

Field Structure ::=

```
<FieldName> is a (Field | KeyField)
    [owned by <ModuleName>]
                        // if not defined then field is global and put in the 'field' package
    [(<SqlName> | <SqlPrefix>)]
                        // SqlPrefix is for Group Field only; SqlName not valid on a Group Field
                        // Every duplicate group field type instantiation in a Business Class or GroupField requires a 'prefix is'
                        // predicate. This lets us add the prefix to all the group field names. Currently, the kludge for the prefix
                        // on a GroupField instantiation in a Business Class is sql prefix is XXX
    [<ClassicName>]
    [<DefaultLabel>]
    [text searchable]
    [scannable]
    [holds pii]
    [default filter operator is (contains | starts with | equals)]
                        // valid only on a SimpleField (not a GroupField or ArrayField) and alpha field types

    [extends <FieldName>]
        [member of <extended FieldName> peer group]
                        // 'extended FieldName' must be a super key field of this key field – that is, this key field must extend it.
                        // A 'peer group' of KeyFields is a set of KeyFields that are all surrogates of each other where the surrogate
                        // values are all identical. If ICCompany extends GLCompany and POCompany extends GLCompany, and if ICCompany
                        // and POCompany are members of the GLCompany peer group, then ICCompany acts as a surrogate for POCompany,
                        // where the POCompany value is equal to the ICCompany value.
        [(delete cascades | delete ignored)]
                        // valid on a KeyField with a business class only - defaults to delete restricted
        [name is <Literal>]
                        // If a field extends another field, it cannot specify a new representation. Both KeyFields and Fields can be
                        // extended. An extended KeyField typically specifies an additional business class that contains the particular
                        // instances of the new KeyField although this is not required. An extended Field typically sets some context
                        // variable of the base Field.
                        // An example of a KeyField being extended with a new business class is Company. Company might be extended
                        // as a new field called APCompany with a business class of ApCompany. If the business class for Company is
                        // Glsystem, each instance of an ApCompany must also be an instance of a Glsystem, whereas each instance of
                        // a Glsystem might not be an instance of an ApCompany.
                        // An example of a KeyField being extended without a new business class being defined is PersonnelCode.
                        // PersonnelCode has a Context variable called PCodeType and it might be extended as a new field called
                        // LocationCode with PCodeType set to PCodeType.Location (one of its states).
                        // An example of a Field being extended is CurrencyValue. CurrencyValue has a context variable called IsRate
                        // and might be extended as a new field called CurrencyAmount with IsRate set to false.
    Representation
    Ontology
    Patterns
    Display Fields
    Surrogate For
    Context Fields
```

The syntax definitions for the following sections in the Field structure are the same as their counterparts in the Business Class structure. Refer to the Business Class Definition section for their syntax.

```
    Transient Fields        // valid on group or array fields only
    Local Fields            // valid on group or array fields only
    Derived Fields          // valid on group or array fields only
    Conditions              // valid on group or array fields only
    Cube Relations          // valid on group or array fields only
    Relations               // valid on group or array fields only
    Rule Blocks
    Field Rules
```

Representation

```
[<FieldRepresentation>]
                        // A KeyField might not have a representation. This is useful when the KeyField is the symbolic key of a business
                        // class that implements the Specialization pattern. The field can still be put as a Persistent field in a business class
                        // to dynamically reference the appropriate related Specialization.

FieldRepresentation ::=
    ( <SimpleField>
    | <GroupField>
    | <ArrayField>
    )

SimpleField ::=
    type <DataDefinition>
        [precision is <RelatedValue>]
            [round to precision]
    [<States>]          // <States> is not indented under type because indentation implies some form of ownership or detail declaration.
                        // In this case, <States> describes the Field's defintion and not the Representation's defintion.

GroupField ::=
    Group Fields
        <FieldName> [<DataDefinition>]
            [<States>]
            [<SqlName>]
            [<ClassicName>]
            [<ClassicNameForField>]
            [<SqlNameForField>]
            [<DefaultLabel>]
            [text searchable]
            [scannable]
            [encrypt]
            [holds pii]
            [enable alternate document location]     // valid for BinaryDocument, BinaryObject, CSVText, JSONObject,
                [document is <RelatedField>]         //    RichText, Text, TextDocument, and XMLDocument fields only
                [existence is <RelatedField>]        // Must be a Boolean type field
            [disable Auditing [when in background]]          // Field will not be audited
            [disable EffectiveDated]                         // Field will not participate in EffectiveDated pattern
            [enable EffectiveDated]                          // Field will participate in EffectiveDated pattern
            [automate context]
            [disable surrogates]
            [allow images]                                   // valid for RichText fields only
            [(delete cascades | delete ignored)]
            [translatable]                                   // allows for data translation for this field
            [restricted]                                     // cannot be used as field on UI
            [protected]          // application-controlled field; cannot be updated from the UI, webservices, or spreadsheet
            [precision is <RelatedValue>]
                [round to precision]
            [primitive type is <RelatedValue>]// must be of  type 'PrimitiveType'; cannot be Text or BinaryDocument/Image
            [primitive size is <RelatedValue>]
            [primitive decimal size is <RelatedValue>]
            [create value is blank if no entry]
                        // If a field is added after a record is created, typically the first value in the audit log applies as the value from
                        // the point of create. With this pattern, the value from create will be considered blank
            [as of <RelatedValue>]
            [within <RelatedValue>]
                [<Message>]
            [[exact] version is (<RelatedValue> | latest)]
            [data area is <RelatedValue>]                    // valid only when stored in an environment busclass
```

```
            [is (condition | related (link | value)) for <RelatedValue>]
                [(dimensions [with attributes] | measures) only]   // RelatedValue must be a BusinessClass name
            [<TextVariables>]
            [document template [for <BusinessClass>]]
                        // if within a Group then just 'document template' can be specified generically. This will require
                        // that wherever this group field is used a specific document template for BusClass is specified
            [store as BusinessObjectReference]
                [<SqlPrefix>]

ArrayField ::=          // ArrayField is valid on both keyfield and fields
    <FieldName> [<DataDefinition>]
        occurs (<Literal> | sizeofarray <ArrayField> | unlimited) times
                        // unlimited is valid for local fields only
```

Ontology

```
// valid on key fields only
    [stereotype is <Stereotype>]
    business class is <BusinessClass>
                        // This is the business class that contains the instances of this KeyField.
                        // This business class must specify this KeyField as its symbolic key.
        [existence not required]

    Context
                        // The ontological context defines the affordance structure. Typically there is only a single KeyField in an
                        // affordance structure. If the single Keyfield itself has an affordance context KeyField, that KeyField is also
                        // implicitly part of this KeyField's affordance structure. If there are two KeyFields in an affordance context, it
                        // means that the second one is not afforded by the first one, therefore there is a dual affordance structure. The
                        // first key can be a surrogate for the higher level context of the second key.
        <KeyField>
            [version is (latest | exact)]
                        // A versioned Context field will automatically have its version field included in the ontology unless
                        // 'version is latest' is specified
            [value is <RelatedValue>]
                        // A Context key can be set to a specific value. Typically this only happens when the key is extending another
                        // key that has this Context key. An example is BargainingUnit, which extends Pcode and sets Pcode's
                        // PcodeType context to 'BU'.
            [delete ( restricted                    // default
                    | cascades
                    | ignored
                    )
            ]
            [disable surrogates]
            [context of <FullFieldName>]…
                        // This can be used to disambiguate which field on this business class should be used as the Context field for this KeyField
            [optional]  // If one of the Context Keys is optional, there must be a higher level Context Key specified even though
                        // the first Context Key implicitly refers to it. This makes it clear that this KeyField can be afforded at both
                        // levels and allows for the delete rule of the higher Context Key to be specified. This higher level Context
                        // Key that is not optional is the group level key; this is the key level at which effective dating is done when
                        // effective date as a group is specified in a Specialization pattern.
            [data area is <RelatedValue>]           // valid only when in a 'stored in environment' busclass
            [(enable | disable) surrogate context]
                        // enable is the default
                        // If a context KeyField has a surrogate, by default this surrogate field is put on the business class
                        // and a secondary index is created so that this field can be validated from either the primary context
                        // or the surrogate context.
                        // If a Key Field extends another key field the default name of that key field is the highest level field name in the
                        // extends chain. This can be overridden with a 'name is' clause. I would like to nuance this situation such that when
                        // a key field extends another key field but has a context value set, then that field should have a default 'name is' of the
```

```
                    // key field name itself.
        [within <RelatedValue>]
            [<Message>]
                    // valid on a KeyField only; <RelatedValue> must be a field referenced via a OTM relation that defines a set of KeyFields
                    // of this KeyField's type. <Message> is an optional error message.

Stereotype ::= ( Agent
               | BusinessPolicy
               | Document
               | DocumentAccounting
               | DocumentDistribution
               | DocumentFulfillment
               | AccountingTransaction
               )
                    // Stereotypes are grouped into two classes: transaction and non-transaction. The Transaction stereotypes are
                    // Document, DocumentDistribution, DocumentFulfillment, and AccountingTransaction.
```

Patterns

```
[disable StaticTranslations]
                    // All <Message> definitions will not participate in Translation processes
                    // There will be no <Field>Bundle.properties file for the BL or the UI in the resource language packs
                    // Conversion of <LiteralMessage> to resource string will still take place

[implements AccessInitializer]
    initializer is <FieldName>
                    // Valid for Group and Array fields only
                    // FieldName must be a DerivedField within this Field

[implements MutuallyExclusive]
    field determiner is <RelatedField>
        <StateName> field is <FieldName>…
                    // valid in a GroupField only. This pattern causes the GroupField to behave as though it is only a
                    // single field that has a dynamic type. The single field within the Group Field is determined by the
                    // defined 'field determiner' <RelatedField> which must be a state field.<StateName> must be a defined
                    // state on <RelatedField>. All states on RelatedField must be specified.
                    // For some unknown reason MutuallyExclusive fields get a default delimiter set (see CompactFormat).

[implements FieldRange]
    from is <FieldName>
        [exclusive]
    to is <FieldName>
        [exclusive]
                    // This pattern is valid in a GroupField only. A from value of blank means low-value and a to value of blank
                    // high-value. To must always be greater than from. Inclusive is the default.This designation allows
                    // the 'within' operator and the 'overlaps' operator to be used with this field.

[implements CompactFormat]
    [delimiter is <Literal>]
    [format fields based on primary form]
                    // This pattern allows a GroupField or an ArrayField to be entered as a single string, each field
                    // being delimited by the designated delimiter. An implication of this is that the delimiter is not
                    // allowed as a character within any of the member or occurring fields. The default delimiter is a double-hat '^^'.
                    // The format (order, visibility, and so on) of the fields can be based on the primary form

[implements UserDefinedStates]
                    // This pattern treats this key field as though it has a list of States – only the state values are not hard-coded
                    // but rather defined within the instances of the business class. There are two key implications of this: one is that
                    // the number of instances is expected to be small and the field should be treated just as a hard-coded field
```

```
                        // that has states on it is in the UI – that is – it should use a drop-down select to allow the user to choose a
                        // value rather than a larger select window.

    [implements ArrayHierarchy]
                        // This pattern causes an Array KeyField to be treated as a Hierarchy of Keys
                        // This is valid only on an Array KeyField and currently all Array KeyFields must implement this pattern.
                        // When a node in an Array Hierarchy is deleted all sub-nodes will be deleted as well – it is an implicit
                        // delete cascades.

    [implements ParentHierarchy]
        [parent field name is <FieldName>]
            [<ClassicName>]
        [descendants are <RelatedLink>]
                        // RelatedLink must point back to this BusinessClass and end in a OTM relation
                        // A ParentHierarchy is a Hierarchy managed by having a 'parent field' (which is a KeyField reference of
                        // this symbolic key) put on the business class that this symbolic key is defined for.
                        // The parent field name defaults to 'Parent<KeyField>'.
                        // When a node in a Parent Hierarchy is deleted all sub-nodes will be deleted as well – it is an implicit
                        // delete cascades.
                        // descendants operator can be optimized by overridding it with 'descendants are…' - this requires you to create a 'shadow'
                        // table that flattens the structure

    [implements Versioning]
        version field is <FieldName>
            [<ClassicName>]
                        // This pattern causes the symbolic key's business class to be versioned using the designated version field.
                        // The version field is often of type Date but can also be other field types, for example a Number.
                        // A field of type FieldName will be added to the business class and also be put at the bottom of the
                        // primary index.

    [implements Specialization]
        [control valid key values]
                        // When the valid key values are controlled then there must exist a key value at the non-optional
                        // key level before it can be specialized. If the highest key value is deleted then all of the specializations
                        // are also deleted (delete cascades). Delete restricted and delete ignored are not supported.
                        // If this is not specified then the key values of the specializations are uncontrolled. A key value can
                        // be added at some specialization level without a higher level value being present.
        [effective date as a group]
                        // Valid only when EffectiveDated pattern is implemented.
                        // This option causes the set of specializations to be effective dated as a group.
        [set retrieval is inclusive]
                        // Exclusive is the default. Inclusive means that the higher level 'specializations' are included as part of the
                        // lower level specializations. An example of this is JobQualifications. The company level qualifications are
                        // considered to be part of the job-specific qualifications.
                        // If any Context Key is optional, the Specialization pattern can be specified. The specialization pattern
                        // allows a single reference to the KeyField to search the business class for the appropriate business object.
                        // The appropriate business object is the one that is the most 'specialized' based on what keys are optional.
                        // The least specialized object is one where all the optional keys are blank. The most specialized object is
                        // one where all the optional keys are filled in. It searches most specialized to least specialized. If the optional
                        // keys are strictly hierarchically related based on their ontology, the search algorithm blanks optional key
                        // values starting from the lowest optional key and moving toward the highest optional key. If the optional
                        // keys are not hierarchically related, the search algorithm will similarly work from lowest to highest key,
                        // however it will put values back into the lower keys. For example, given keys k1, k2, k3 with k2 and k3 optional
                        // and not hierarchically related, the search pattern is (k1, k2, k3), (k1, k2, blank), (k1, blank, k3),
                        // (k1, blank, blank).
```

Display Fields

```
// valid on key fields only
    <RelatedField>      // These are the 'description' or 'name' fields for this key. They are the 'human' names we give things, such
                        // as company name, employee name, and so on
```

Surrogate For

```
// valid on key fields only
     <FieldName> value is <RelatedField>

// The Context Fields section declares a new 'non-ontological' context field. Non-ontological context fields are not required to be in the context. They are also
// evaluated newly each time the field is used. Context rules are not automatically arranged for execution as are Field Rules. The order they are in is the order
// in which they are executed.

// Special considerations for field addressing, or what does the '.' operator do when used with a KeyField, GroupField, or ArrayField.
// On a KeyField – the '.' operator automatically addresses all the elements on the business class that the KeyField is centered on.

// If KeyField APCompany is a subtype of another KeyField Company then in order to address fields on the business class Company from APCompany you
// say 'APCompany.Company.FieldName'
// Similarly in order to address fields on APCompany from Company you say Company.APCompany.FieldName.

// On a GroupField – the '.' operator or any Field reference in a business class can reach through a GroupField or an ArrayField and address a Field
// on the GroupField if the reference results in an unambiguous target. If Address is a GroupField that has a GroupField called PostalArea, which in turn
// contains City, State, and Zip, the Reference Address.City reaches through the GroupField PostalArea to find its target. It also reaches through an
// ArrayField. It does not reach through KeyFields to the KeyField's business class. If Address is on the business class Vendor, any rule on Vendor can
// directly address 'City' and it does reach through both Address and PostalArea to find 'City'.
```

Context Fields

```
// These are 'non-ontological' context variables
     <ContextField>…

     ContextField ::=
          <FieldName> [<DataDefinition>]
               [[exact] version is (<RelatedValue> | latest)]
               [<ContextRule>…]

     ContextRule ::=
          <RelatedField> <AssignmentOp> <RelatedValue>                    //Assignment Rule

          default to <RelatedValue>…     // Multiple fields can be put on a default. If the first one is blank the second one is used and so on.
                                         // Only the last one can be a literal value.
          constraint <Condition>
               <Message>

          required
               [<Message>]               // defaults to 'Field is required'

          if <Parens><FieldName> <ContextOperator><Parens>
               <ContextRule>
          [else
               <ContextRule>]

     ContextOperator ::=   ( in context
                           | entered
                           )
```

Business Class Definition

// Business Class files have an extension of .busclass

Business Class Structure ::=

```
<BusinessClass> is a BusinessClass
    [deprecated]
    [owned by <ModuleName>]            // If not defined, this defaults to the module it is under
    [<ClassicPrefix>]                  // Required for use in Cobol programs
    [<SqlName>]
    [<RpgName>]
    [<ClassicName>]
    [<DefaultLabel>]
    [framework type is    ( ConfigParam
                          | LPLConfiguration
                          | ProcessFlow
                          | ReplicationSet
                          | Security)]
    [untranslatable]                   // prevents any text or data from being translated
    [subject is <Subject>]             // default subject for all actions
    [stored in [base] environment]     // Store data in the environment database, IJF framework only
    [stored in product line]           // Store data in the environment product line database (vs standard data area DB)
    [store using <ClassName>]          // ClassName that implements persistence, IJF framework only
    [contains environment data]        // Environment data is not copied in daexport, IJF framework only
    [disable data area copy]           // Prevent data from being copied in daexport, IJF framework only
        [preserve target data]         // default is to delete target data
    [representative text is <Message>]// Text that is the human readable form of the key to this business class (replaces Display Fields)
    [representative image is (<RelatedValue> | <Icon>)]          // Image that represents an instance of this business class
        [display as (portrait | photo | full) image]             // default is 'display as portrait'
        [missing image is (<ImageName> | random background)]
            [foreground text is <Message>]
    [(restrict | enable) actions on lists]                                  // default is 'enable'
    [default Alpha filter operator is (contains | starts with | equals)]    // default is 'contains'

    Suppress Warnings
    Ontology
    Patterns
    DataSource Mapping
    Persistent Fields
    Transient Fields
    Dimensions
    Measures
    Cube Links
    Context Fields
    Field Groups
    Audit Index Fields
    Local Fields
    Derived Fields
    Conditions
    Text Search Fields
    Cube Relations
    Relations
    Form Invokes
    Matrix Forms
    Sets
    Field Rules
    SubType <RelatedCondition> Field Rules
```

Commit Rules
Audit Entry Rules
Create Rules
Create Exit Rules
Update Action Rules
Delete Rules
Attach Rules
Parent Attach Rules
Dynamic Creation Rules
Rule Blocks
StateCycles
Actions

Suppress Warnings

```
( set size on <SetName>
| all set size warnings
)…
```

Ontology

```
symbolic key is <KeyField>                    // can have multiple symbolic keys; these are synonyms
    [name is <FieldName>]                     // overrides the name of the symbolic KeyField on this class
    [classic set name is <Literal>]           // sets the classic name for the primary set that is generated
    [sql set name is <Literal>]               // sets the sql name for the primary set that is generated
    [not indexed]                             // do not create a physical index for the symbolic key
    [<SqlName>]
    [<ClassicName>]
    [<ClassicNameForField>]
    [<SqlNameForField>]
    [protected]       // application-controlled field; cannot be updated from the UI, webservices, or spreadsheet
    [name for <context FieldName> is <FieldName>]
                      // <ContextField> must be a context field within <KeyField>. The context fields within a KeyField
                      // automatically get put on the business class. The default name is the highest level type name of the
                      // context field. If APCompany is the context field and APCompany extends Company, the name is Company.
                      // This phrase allows the field name to be explicitly defined.
    [type for <context FieldName> is a[n] <KeyField>]
                      // This overrides the type of the Context Field. The new type must be a subtype of the original type, ADCompany
                      // versus Company. This 'symbolically keys' the business class. A business class can be keyed both symbolically
                      // and relatively. A business class that is not symbolically keyed is considered to be 'relatively' keyed.
                      // When a KeyField is defined to be the symbolic key to a business class, the Ontology Context variables
                      // are replicated on the business class. When a KeyField is simply used on a business class, the Ontology
                      // Context variables must be resolved within the context of the business class or a compile error occurs. The
                      // exceptions to this are those Ontology Context variables that are designated as optional.

part of <BusinessClass>
    [connect via UniqueID [only]]
                      // BusinessClass must NOT implement the LazyUniqueID pattern to use connect via UniqueID
    [delete ( cascades                // default
            | restricted
            | ignored
            )
    ]
    relative key is <FieldName> [<DataDefinition>]
```

```
[<States>]
[<SqlName>]
[classic set name is <Literal>]          // sets the classic name for the primary set that is generated
[sql set name is <Literal>]              // sets the sql name for the primary set that is generated
[not indexed]                            // do not create a physical index for the symbolic key
[<ClassicName>]
[<ClassicNameForField>]
[<SqlNameForField>]
[<DefaultLabel>]
[protected]  // application-controlled field; cannot be updated from the UI, webservices, or spreadsheet
[(delete cascades | delete ignored)]
[[exact] version is (<RelatedValue> | latest)]
                // This directive specifies the particular version of a versioned key field to retrieve.
                // If the version field on the key field is of type Date, TimeStamp, or Time this will default to the current date,
                // current timestamp, or current time otherwise it will default to the latest version.
                // If there is no exact match on the version it will return the closest version before the version specified.
                // If 'exact version is' is specified there must be an exact match on the version.
                // If 'latest' version is specified, it will find the last version even if that is a future version.
[data area is <RelatedValue>]            // valid only when in a 'stored in environment' busclass
[name for <context FieldName> is <FieldName>]
                // <ContextField> must be a field in the context field chain of <BusinessClass>. The context field chain of
                // <BusinessClass> automatically gets put on this business class. The default name is the highest level type name of the
                // context field. If APCompany is the context field and APCompany extends Company, the name is Company.
                // This phrase allows the field name to be explicitly defined.
[type for <context FieldName> is a[n] <KeyField>]
                // This overrides the type of the Context Field. See above definition with symbolic key for full description
[text searchable]
                // Only one relative key is allowed. This keys the business class relative to another business class.
                // The 'relative key' construct will automatically put the <BusinessClass> SymbolicKey
                // field, including its Ontological Context fields, on this business class unless connect via UniqueID only
                // is specified. It also automatically puts a one-to-one or one-to-many relation on <BusinessClass> pointing back
                // to this business class. It names that relation '<this business class>SetRel'. If <BusinessClass> is itself
                // relatively keyed, a one-to-one required relation will be put on this business class. It names that relation
                // '<BusinessClass>Rel'.
                // The 'parts' of <BusinessClass> can be accessed from <BusinessClass> by specifying '<this business class> set' in
                // a <RelatedLink>
                // The <BusinessClass> can be accessed from <this business class> by specifying '<BusinessClass>' in a
                // <RelatedLink>
```

Patterns

```
[disable Auditing]                  // By default non-'Classic' Business Classes implement the Auditing pattern
                                    // BinaryDocuments and BinaryObjects do not participate in this pattern

[implements LightweightAuditing]    // Initial create does not create the log; it is instantiated on the first update or delete

[disable AuditIndex]                // By default an AuditIndex (the 'snapshot' table) is created when a BusinessClass is Audited and
                                    // AsOfDateProcessing is not disabled. The AuditIndex optimizes as of date queries at the expense of
                                    //  creates and updates.

[implements EffectiveDated]
    [bypass integrity rules]

[disable EffectiveDated]            // By default non-'Classic' Business Classes implement the EffectiveDated pattern
                                    // Also by default BinaryDocument and BinaryObject fields do not participate in this pattern. They can
                                    // be made to participate in this pattern by specifying 'enable EffectiveDated' on the field definition

[implements AuditLogEntryActions] // force AuditLogEntry record create for actions (overrides config param)
```

```
[implements OnceUsedNeverDelete]

[implements DeleteFlag]            // Valid only for business classes that specify "store using"

[disable RetroactiveEffectiveDating]

[disable UniqueID]

[disable AsOfDateProcessing]

[disable DataTranslations]         // This disables all data translation for this business class.

[disable StaticTranslations]       // All <Message> definitions will not participate in Translation processes
                                   // There will be no <BusinessClass>Bundle.properties file for the BL or the UI in the resource language
                                   // packs. Converstion of <LiteralMessage> to resource string will still take place

[implements HistoryCorrection]     // Allows past audit log entries to be adjusted

[implements FutureCorrection]      // Allows future entries to be adjusted

[implements ForceUIRefreshOnStale]

[implements IncrementalReplication]            // if no indicator field is specified then must have a change stamp meaning
    [indicator field is <FullFieldName>]       //   that it must be audited or implements UpdateStamp
        replicate when <Value>
            then (set to <Value> | invoke <ActionName> [<FullStateName>])

[disable IncrementalReplication]   // disables IncrementalReplication if implemented at the Product Line level

[implements InlineUserFields]      // Puts user field values directly on the business class in a VarChar field
    size is <Number>               // Must define how big the field is that contains all the user field values
                                   // Field will contain user field name plus actual value

[implements StaticJava]            // This pattern generates a StaticJava DB interface class
    [has attachments]              // Generates <BusClass>_URL and <BusClass_Comment tables

[implements InMemoryCache]         // This pattern stores and retrieves business class instances from an in-memory cache.
                                   // The cache is valid only within a process and records can only be retrieved through the KeyField.

[implements WorkFile]              // This causes the business class to be treated as a WorkFile

[implements ForeignTable]          // This pattern by default implements the ReadOnly pattern and the LazyUniqueId pattern
    [allow updates]

[implements ReadOnly]              // The ReadOnly pattern can be inhibited by specifying 'allow updates'

[implements LazyUniqueID]          // All business classes automatically get a UniqueId put on them to support the ability to address any
                                   // LPL based business class directly through a UniqueId. If the business class is not created through
                                   // the IJF runtime but is rather an 8.1 or third party database table then it needs to implement
                                   // this pattern. This pattern will generate an outboard file and the appropriate logic in the business
                                   // class to allow the IJF runtime to access the outboard table via a UniqueId.

[implements S3ClassicTable]        // This pattern by default implements the ReadOnly pattern and the LazyUniqueId pattern
    [allow updates]                // The ReadOnly pattern can be inhibited by specifying 'allow updates'
    [implements UniqueID]          // By default UniqueID is disabled and LazyUniqueId is enabled
    [has attachments]              // Comment type attachments are accessed through a Text field name <AttachmentField>
        [<attachment FieldName> is comment of type <Literal>…]
                                   // All comments of a particular comment type are concatenated into a single Text field.
                                   // Also disables Auditing and EffectiveDated
```

```
    [implements S3ClassicAttachments]
        [<attachment FieldName> is comment of type <Literal>…]

    [implements M3ClassicTable]
        [allow updates]                 // This pattern by default implements the ReadOnly pattern and the LazyUniqueId pattern
                                        // The ReadOnly pattern can be inhibited by specifying 'allow updates'
                                        // Also disables Audting and EffectiveDated

    [implements Resequence on <FieldName>]
        new sequence field is <FieldName>
        set is <SetName>
                                        // 'FieldName' must be numeric. Resequence field and new sequence field must be different fields.
                                        // SetName must contain the Resequence field. Whenever the new sequence field is entered with a
                                        // different value from the resequence field the record will be moved to the new sequence field position
                                        // and then the whole set as defined by SetName will be resequenced
                                        // Typically new sequence field would be a Transient Field
                                        // This pattern will also automatically generate an autosequence rule if the sequence field as long as
                                        // it does not already have an autosequence rule manually defined
                                        // When doing mass import using an update action the new sequence field can be set to -1 to prevent
                                        // resequencing, which will significantly improve performance

    [implements DynamicCreation]        // Causes an object to be created automatically if it does not exist when another business object
                                        // 'attaches' to it or if an Update type action is called. The creation can be constrained or special
                                        // action can be taken upon dynamic creation by putting rules in the Dynamic Creation Rules section.

    [implements AnalyticCube]
        [label is <Message>)]           // if not blank, this will be used as the cube name caption
        [columnar only]
        [(pre-calculate totals | dynamically calculate totals)]    // default is 'pre-calculate totals'
        [(allow write to summary level | write to base level only)]
                                                                   // default is 'allow write to summary level'
        [refresh using (audit log | message queue)]                // default is audit log unless auditing is disabled
        [beginning balance is period zero]                         // period operators will use period zero for beginning balance
                                        // the default is to calculate the beginning balance as the sum of all the prior periods
                                        // this requires the designer to explicitly place the beginning balance in period zero
        [Instance Selection]
            where <Condition>           // limits what fact rows are put in the cube

// A ChildHierarchy is a Hierarchy defined by its children. It can have multiple parents, which means that the hierarchy operators parent, siblings,
// ancestors, and ascendant are not valid. When used as a Dimension 'child set is…' is required and 'top node when…' is highly recommended.
    [implements ChildHierarchy]
        [top node when <SimpleCondition>]
        [leaf node when <SimpleCondition>]
        ( children are <RelatedLink>
        | child set is <RelatedLink>                          // required if this KeyField is used in a Dimension
            child is <KeyField>                               // must be a KeyField to this business class
            [aggregation percent is <FullFieldName>]          // must be a Percent field
        )                   // The 'children are' RelatedLink must point back to this same business class and end in a OTM relation
                            // The 'child set is' RelatedLink must point to the intermediate business class that determines the children
                            // The 'child is' KeyField must then point back to this business class
        [descendants are <RelatedLink>]                       // similar to 'children are' but for all descendants

    [implements CreateStamp]
    [implements UpdateStamp]

    [implements Proxy for <RelatedLink>]            // must be OTO relation or KeyField
        [Additional Field Mapping]
            related.<FullFieldName> = <RelatedValue>…

    [implements CompoundDocument]                   // must have at least one RelatedLink
        Document Components
```

```
        <RelatedLink>…

[implements BODId]

[implements Encrypted]
    [when <Condition>]              // can reference only mutable non-related fields

[implements ContextualParent]      // extends Context search to ontological parent business class

[implements SurrogateAuditLog for <FullFieldName>]
                                   // causes the audit log of this business class to really be the audit log of FullFieldName
                                   // FullFieldName must be a Text field that contains a serialized DataView
                                   // This is currently only being used for environment business classes

[implements TemplateDriven by <BusinessClass>]
    [completion message is <Message>]          // <Message> can contain text only
    [( create all instances                    // 'create all instances' is the default
     | create instance
        when <Condition>
     )                                         // can reference only mutable non-related fields
    ]                  // For example, Answer is a part of Response and extends Question. Answer is designated as being TemplateDriven by.
                       // Question. Question is a template for adding, updating and deleting corresponding Answer instances. An Answer instance.
                       // cannot be created without a corresponding Question instance.
                       // This pattern works with a <Template> definition within a <NavigationDefinition> or a <PanelDefinition>
                       // If create all instances is specified, an Answer instance will be created for every Question in the template list..
                       // If the create is conditional, an Answer instance will be created only if the condition evaluates to true. If the condition.
                       // is false any existing corresponding Answer instance will be deleted.

[implements <DataSourceType> DataSource]
                       // This pattern defines an alternate data source to satisfy the business class contract. This means the CRUD pattern
                       // and additional actions are realized by an external data source.

[implements ExtendsExternalData]
    [using <ClassName>]            // ClassName is a JavaClass that provides access to the external data
                       // This pattern allows for the extension of an external data source. With this pattern external data is viewed as a business
                       // class and the user is able to logically add to or change that data. The actual underlying data is not changed. The
                       // underlying data might be a property file on disk or some other flat file format.
```

DataSource Mapping

```
<DataSourceCRUDType> is <DataSourceAction>
    <InvokeRule>

DataSourceType ::=  ( MI           // MI uses a .mi interface definition to fulfill the contract
                    | WSI          // WSI uses a .wsi interface definition to fulfill the contract
                    )

DataSourceCRUDType ::=  ( set
                        | find
                        | create
                        | update
                        | delete
                        )

DataSourceAction ::= <WebserviceInterface>.<ActionName>
```

Persistent Fields

```
<FieldName> [<DataDefinition>]
    [<States>]
    [<SqlName> | <SqlPrefix>]
    [<ClassicName>]
    [<ClassicNameForField>]
    [<DefaultLabel>]
    [encrypt]                          // causes the field to be encrypted in the database; valid on Alpha fields only
    [automate context]                 // if the field is a KeyField this will ensure that all context variables are placed on this bus class
    [enable alternate document location]        // valid for BinaryDocument, BinaryObject, CSVText, JSONObject,
        [document is <RelatedField>]            //    RichText, Text, Text Document, and XMLDocument fields only
        [existence is <RelatedField>]           // Must be a Boolean type field
    [allow images]                     // valid for RichText fields only
    [text searchable]                  // means this field will be search text indexed by the database
    [scannable]
    [holds pii]
    [translatable]                     // allows for data translation for this field
    [disable Auditing [when in background]]    // Field will not be audited
    [disable EffectiveDated]           // Field will not participate in EffectiveDated pattern
    [enable EffectiveDated]        // Field will participate in EffectiveDated pattern (valid on BinaryDocument and BinaryObject only)
    [restricted]                       // cannot be used as field on UI
    [protected]                        // application-controlled field; cannot be updated from the UI, webservices, or spreadsheet
    [precision is <RelatedValue>]
        [round to precision]
    [primitive type is <RelatedValue>]// must be of type 'PrimitiveType'; cannot be Text or BinaryDocument/Image
    [primitive size is <RelatedValue>]
    [primitive decimal size is <RelatedValue>]
    [create value is blank if no entry]
                        // If a field is added after a record is created, typically the first value in the audit log applies as the value from
                        // the point of create. With this pattern, the value from create will be considered blank
    [disable surrogates]
    [context of <FullFieldName>]…
                        // This can be used to disambiguate which field on this business class should be used as the Context field for this KeyField
    [(delete cascades | delete ignored)]
                        // <FieldName> must be a KeyField. The default delete rule for a KeyField is delete restricted
                        // Use of a KeyField will cause an OTM back relation to be automatically generated
    [as of <RelatedValue>]
                        // <FieldName> must be a KeyField, <RelatedValue> must be a TimeStamp or a Date
                        // this directive will retrieve the business class that this keyfield points to 'as of' the specified date and time
                        // an implication of this is that the one-to-many relation from the keyfield's business class back to this
                        // business class will be 'delete ignored'. If <RelatedValue> is a Date then the last entry on that date will be retrieved.
    [within <RelatedValue>]
        [<Message>]
                        // valid on a KeyField only; <RelatedValue> must be a field reference via an OTM relation that defines a set of KeyFields
                        // of this KeyField's type. <Message> is an optional error message.
    [[exact] version is (<RelatedValue> | latest)]
                        // This directive specifies the particular version of a versioned key field to retrieve. If the version field on the
                        // key field is of type Date, TimeStamp or Time this will default to the current date, current timestamp or current time
                        // otherwise it will default to the latest version. If there is no exact match on the version it will return the closest version
                        // prior to the version specified.If 'exact version is' is specified then there must be an exact match on the version.
                        // if 'latest' version is specified then it will always find the last version even if that is a future version.
    [data area is <RelatedValue>]
                        // valid only when in a 'stored in environment' busclass
    [is (condition | related (link | value)) for <RelatedValue>]
        [(dimensions [with attributes] | measures) only]    // RelatedValue must be a BusinessClass name
    [<TextVariables>]
                        // Text variables can be defined for any Alpha type field. This allows the user who is entering data in this
                        // field to put in variable substitution syntax that references these defined variable names. The syntax to
                        // use a text variable is '{TextVariable}'. Notice the curly braces special syntax that
                        // surrounds the TextVariable name. TextVariables can be one-to-many valued. If the variable starts at
```

```
                        // the beginning of a line and ends with a new line character then it will be copied down – each OTM being a new line.
                        // If not then it will act as though it is in paragraph mode.
                        // A value does not have to be ascribed to a TextVariable definition. This will mean that no substitution ever
                        // takes place. This is useful when defining text fields that are used as templates for other text fields. The only
                        // issue is that the designer needs to make sure the variable name match on both the template and the target.
                        // If a text variable uses a Context Field in its RelatedLink then the value of the Context Field must be specified
                        // as such '{TextVariable:<ContextFieldValue>}'
    [document template for <BusinessClass>]
                        // This field must be of the type 'BinaryDocument'. A document template is a 'rtf' file with standard
                        // Word Doc replacement variables which must be of LPL syntax. When the 'document' TypeOperator is used
                        // on this field the 'rtf' is sent through a variable replacement transformation resulting in another 'rtf' file
                        // that can be assigned to a BinaryDocument field. This allows for Word Documents to be generated within LPL.
    [store as BusinessObjectReference]          // valid on a KeyField only
        [<SqlPrefix>]
                        // This is required when trying to assign this value from an asynchronous operation. LawsonClassicTransactions
                        // are an example of an asynchronous operation. When creating a new record via an asynchronous operation
                        // only the BusinessObjectReference of the record is immediately available and thus only that aspect of the
                        // keyfield can actually be stored. A BusinessObjectReference is internally a GroupField and may have sql
                        // naming conflicts. The 'sql prefix' statement can be used to overcome any naming conflict.
    [Valid Business Classes]                    // valid on a BusinessObjectReference only
        <BusinessClass>…

<FieldName> is a snapshot of <RelatedField>
    [when <Condition>]
                        // snapshot sets and keeps this field the same as RelatedField only when this business object is being updated.
                        // This can optionally be done based on a Condition

(<FieldName> | <FieldGroupName>) from <BusinessClass>

(audit fields | <ActionAttribute>…)             // valid only when Auditing and EffectiveDated are disabled

TextVariables ::=
    Text Variables [<Parens>locale of <RelatedValue><Parens>]  // RelatedValue must be an IsoLocale
        <TextVariable> [value is <RelatedValue>]…
                        // A TextVariable can be placed in a Text field surrounded by curly braces as such, {<TextVariable>}. When the
                        // TypeOperator text is used on this field, the TextVariable will be replaced in the Text field with the designated
                        // RelatedValue. RelatedValues can be directly used for variable replacement by prefixing the related value with
                        // lpl:, making the syntax {lpl:<RelatedValue>}. This allows for UserFields and other dynamic LPL constructs
                        // to be used at runtime without having to have a TextVariable.

AggregateOp ::= ( aggregate
                | instance count
                )
                        // The aggregate operator can be used only in a business class that is an aggregate of the
                        // an aggregated <BusinessClass>. If the operator instance count is used, .<FullFieldName> cannot be
                        // /specified.If the operator aggregate is used, .<FullFieldName> must be specified and must be a field in the
                        // aggregated <BusinessClass> that is capable of being aggregated. It must be some form of numeric field.
                        // It can be a group field with non-aggregatable fields in it if there is at least one aggregatable field.
```

Transient Fields

```
// Mutable but not persistent
    <FieldName> [<DataDefinition>]
        [<States>]
        [<DefaultLabel>]
        [scannable]
        [holds pii]
        [precision is <RelatedValue>]
            [round to precision]
```

```
[primitive type is <RelatedValue>]// Must be of type 'PrimitiveType'; cannot be Text or BinaryDocument/Image
[primitive size is <RelatedValue>]
[primitive decimal size is <RelatedValue>]
[as of <RelatedValue>]
[within <RelatedValue>]
    [<Message>]
[disable surrogates]
[[exact] version is (<RelatedValue> | latest)]
[data area is <RelatedValue>]                  // valid only when in a 'stored in environment' busclass
[is (condition | related (link | value)) for <RelatedValue>]
    [(dimensions [with attributes] | measures) only]    // RelatedValue must be a BusinessClass name
[<TextVariables>]
[document template for <BusinessClass>]
[store as BusinessObjectReference]             // valid on a KeyField only
[derive value from <RelatedValue>]             // value when field is not entered
```

Dimensions

```
<RelatedField>          // must be a KeyField or a StateField or designated as a 'period'- can be only a one-to-one related field
    [dimension name is <Literal>] // default is business class prefix plus field name
                                  // if the default exceeds the name limit of 50 characters, you must specify a name
    [label is <Message>)]         // if not blank, this will be used as the cube dimension name caption
    [suppress dimension when <SimpleCondition>]
    [caption is (representative text | <Message>)]    // default text to display in cube; defaults to representative text
    [default total node is <RelatedField>]     // If a default total node is not specified, an All node , which aggregates all the
                                               // dimension values, is generated and used when no dimension value is specified.
    [top node is <RelatedField>]               // top or 'start' node when dimension is parent or child hierarchy
    [disable hierarchy aggregation]            // by default all hierarchy dimensions will aggregate based on the hierarchy
    [is a <PeriodType> period dimension [with year of <FullFieldName>]]
                        // daily periods only roll up into months based on a Gregorian calendar
                        // if the RelatedField is a date then it must be a daily period dimension and does not need a year designation
                        // 'with year…' is required on all non date-based periods
        [current year is <RelatedField>]
        [current period is <RelatedField>]
        [current date is <RelatedField>]       // valid on a date-based period only; if no current period is defined,
                                               // the current period will be determined based on the 'as of' date
        [period set is <RelatedField>]         // valid on a date-based period only

// The following specifications are required for PeriodOperators to work with the 'variable' PeriodType
// Condition and FullFieldName below are in the context of the dimension business class
            [default label is <Message>]                  // period label
            [current date period is <FullFieldName>]      // period key field that corresponds to the current date
            [year when <Condition>]                       // true when period is a year
            [quarter when <Condition>]                    // true when period is a quarter
            [month when <Condition>]                      // true when period is a month
            [week when <Condition>]                       // true when period is a week
            [day when <Condition>]                        // true when period is a day
            [start date is <FullFieldName>]               // Date field that returns period start date
            [end date is <FullFieldName>]                 // Date field that returns period end date
            [parent period is <FullFieldName>]            // period key field for parent period
            [next period is <FullFieldName>]              // period key field for next period
            [previous period is <FullFieldName>]          // period key field for previous period
            [beginning balance is <FullFieldName>]        // period key field for ltd beginning balance
            [ending balance is <FullFieldName>]           // period key field for ltd ending balance
            [beginning ytd balance is <FullFieldName>]    // period key field for ytd beginning balance
            [ending ytd balance is <FullFieldName>]       // period key field for ytd ending balance
```

```
    [valid for measure <FullFieldName>]…
    [(allow write to summary level | write to base level only)]
                                    // default is 'allow write to summary level' unless changed at the AnalyticCube level
    [Instance Selection]
        where <Condition>           // Valid when RelatedField is a KeyField or a one-to-one related field. The Condition is
                                    // defined in the context of the KeyField / OTO BusinessClass.
    [Attributes]
        [child.]<FullFieldName>…    // FullFieldName is a field in the KeyField's BusinessClass. If the KeyField is a ChildHierarchy
                                    // then fields in the child set BusinessClass can also be referenced using the 'child' data link.

    [Hierarchies]
        <HierarchyName>             // generates a rollup hierarchy using the specified attributes
            [base level (hidden | included)]   // default is included
            [description is <Message>]         // defaults to HierarchyName
            Attributes
                [child.]<FullFieldName>…       // must be a declared Attribute

PeriodType ::= ( monthly
               | weekly
               | daily
               | variable
               )
```

Measure Based Dimensions

```
// A Measure Based Dimension is not an existing field value but rather takes on a value based on each measure. Each measure must specify the value for this field.
<FieldName> [<DataDefinition>]                  // must be a KeyField or a StateField
    [dimension name is <Literal>]               // default is business class prefix plus field name
                                                // if the default exceeds the name limit of 50 characters, you must specify a name
    [label is <Message>)]                       // if not blank, this will be used as the cube dimension name caption
    [caption is (representative text | <Message>)]    // default text to display in cube; defaults to representative text
    [default total node is <RelatedField>]      // If a default total node is not specified, an All node , which aggregates all the
                                                // dimension values, is generated and used when no dimension value is specified

    [Instance Selection]
        where <Condition>           // Valid when RelatedField is a KeyField or a one-to-one related field.
                                    // The Condition is defined in the context of the KeyField / OTO BusinessClass.
    [Attributes]
        [child.]<FullFieldName>…    // FullFieldName is a field in the KeyField's BusinessClass. If the KeyField is a ChildHierarchy,
                                    // then fields in the child set BusinessClass can also be referenced using the 'child' data link.
            [is aggregatable]       // generates a rollup hierarchy over this attribute in the cube

    [Hierarchies]
        <HierarchyName>                         // generates a rollup hierarchy using the specified attributes
            [base level (hidden | included)]    // default is included
            [description is <Message>]          // defaults to HierarchyName
            Attributes
                [child.]<FullFieldName>…        // must be a declared Attribute, then fields in the child set BusinessClass
                                                // can also be referenced using the 'child' data link.
```

Measures

```
<FullFieldName>                 // must be a Persistent or Derived field that is numeric
```

```
[measure name is <Literal>]     // default is business class prefix plus field name; default max is 50, otherwise specify a name
[calculate dynamically]         // valid only when measure is a ComputeField
[<DimensionField> value is <RelatedValue>]
                                // DimensionField must be a Measure Based Dimension
                                // When using a Measure Based Dimension, measure names can be duplicated
[cube rule is <Message>]        // FullFieldName must be a NativeField
[Dimension Based Measures]
    <FieldName>…
        [<DefaultLabel>]
        Dimension Values
            <DimensionField> value is <RelatedValue>…
```

Cube Links

```
// Link to another BusinessClass's cube that automatically extends this cube's measures to include all the measures of the linked cube
    <CubeLinkName>
        link to <BusinessClass>         // BusinessClass must have a cube defined
        [Dimension Mapping]
            related.<DimensionField> = (<DimensionField> | top node)…
```

Context Fields

```
    <ContextField>…
        [<DefaultLabel>]
        [is (condition | related (link | value)) for <RelatedValue>]
            [(dimensions [with attributes] | measures) only]    // RelatedValue must be a BusinessClass name
```

Field Groups

```
// Arbitrary grouping of fields within this business class
    <FieldGroupName>
        [include [(persistent | auditable | derived)] user fields]
        <FullFieldName>…
```

Audit Index Fields

```
// List of fields to add to audit index (snapshot table) by default all fields in Sets are put in the audit index
    <FullFieldName>…
```

Local Fields

```
    <FieldName> [( <DataDefinition>
                | is a <BusinessClass> view
                )]
```

```
    [<DefaultLabel>]
    [value is <RelatedField>]
    [<States>]
    [holds pii]
    [precision is <RelatedValue>]
        [round to precision]
    [as of <RelatedValue>]
    [disable surrogates]
    [[exact] version is (<RelatedValue> | latest)]
    [data area is <RelatedValue>]              // valid only when in a 'stored in environment' busclass
    [is (condition | related (link | value)) for <RelatedValue>]
        [(dimensions [with attributes] | measures) only]    // RelatedValue must be a BusinessClass name
    [<TextVariables>]
    [document template for <BusinessClass>]
    [store as BusinessObjectReference]         // valid on a KeyField only
    [do not save in checkpoint]
```

Derived Fields

```
<FieldName> is a[n] <DerivedFieldType>
    [type (<DataDefinition> | is a[n] <BusinessClass> view | is MessageField)]
        [precision is <RelatedValue>]
                    // type is not required for all derived fields. On some derived fields, such as Compute, the type defaults from the
                    // fields in the compute statement.
    [is (condition | related (link | value)) for <RelatedValue>]
                    // RelatedValue must be a BusinessClass name
    [<SqlName>]
    [<ClassicName>]
    [<DefaultLabel>]
    [restricted]      // cannot be used as field on UI
    [holds pii]
    (<DerivedFieldExpression> | <optionally blank for NativeField and InstanceCount>)

DerivedFieldType ::= ( aggregation of <FullFieldName>
                     | ConditionalField
                     | ComputeField
                     | InstanceCount
                     | StringField
                     | MessageField
                     | LabelField
                     | DerivedField)

DerivedFieldExpression ::=
    aggregation of
        when <Condition>            // useful for cube measures - returns FullFieldName value or 0 on instance depending on condition
    ConditionalField
        <ConditionalFieldControl>
    ComputeField
        <Parens><RelatedValue> [<MathOperator> <RelatedValue>]<Parens>…
    InstanceCount
        [where <Condition>]         // useful for cube measures - returns 1 or 0 on instance depending on condition
    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> | return [<RelatedValue>])…

Condition ::=  <Parens><ConditionNode><Parens>
               [<Conjunction> <Parens><ConditionNode><Parens>…]

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
                  | *
                  | /
                  | ^
                  | %
                  )
```

Conditions

```
<ConditionName>
    [<SqlName>]
    [<ClassicName>]
    [<DefaultLabel>]
    [restricted]      // cannot be used as field on UI
    when <Condition>

Conjunction ::=  ( and
                 | or
                 )

ConditionNode ::= ( <ActorCondition>
                  | <ChangedCondition>
                  | <ComparisonCondition>
                  | <FieldCondition>
                  | <FieldWasCondition>
                  | <IsABusClassCondition>
                  | <RelatedCondition>
                  | <RelationCondition>
                  | <WithinBusClassGroupCondition>
                  )

ActorCondition ::= [authenticated] actor (has role | in group) <Text>
                       // Text is either a role name or an actor group name

ChangedCondition ::= (<RelatedLink> | <RelatedField> | [<RelatedLink>].<FieldGroupName>)
                                changed between <RelatedValue> and <RelatedValue>
                       // 'changed between' RelatedValues must be Date or TimeStamp

ComparisonCondition ::= <RelatedValue> <ConditionOperator> <RelatedValue>
```

```
FieldCondition ::=  [<FieldOperator>](<RelatedField> | <FieldGroupName>)
                        ( entered                   // not yet implemented with <FieldGroupName>
                        | is numeric                // not yet implemented with <FieldGroupName>
                        | is high value
                        | changed
                        | in context                // not yet implemented with <FieldGroupName>
                        | is leap year
                        | is last day in month
                        | has audit change          // valid only in context of an audit entry
                        )
                    // You cannot use a <RelatedField> with 'changed' and 'in context'. These can take a <FullFieldName>.

FieldWasCondition ::= <FullFieldName> was [always] <RelatedValue>
                                    [between <RelatedValue> and <RelatedValue>]
                    // between RelatedValues must be Date or TimeStamp

IsABusClassCondition ::= <RelatedField> is a[n] <BusinessClass>

RelationCondition ::= [(first | no[t])] <RelatedLink> (exists | pending | has future changes)
                    //' pending' is valid only with a BusinessObjectReference field or a KeyField that is designated as
                    // 'store as BusinessObjectReference'. It returns true if the reference is filled out but the record
                    // does not actually exist yet. Currently this can happen only when using the asynchronous AGS or other
                    // async third party interface.
                    // 'has future changes' returns true if there are future effective dated changes from the current as of date

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 valid with a group field that implements the FieldRange pattern as the right operands.
                    // The within operator is valid with TimeStamp, Date, Period (month), Year, and month and week operators
                    //     on both sides as long as the the left operand has more specificity than the right operand.
                    //     So (<TimeStamp> within <Date>) is valid but (<Date> within <TimeStamp>) is not.
                    // The overlaps operator is valid only with two 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
                    //      "Mastering Regular Expressions, 2nd Edition," Jeffrey E.F. Friedl, O'Reilly and Associates, 2002.

WithinBusClassGroupCondition ::= <RelatedLink> [not] within <RelatedField>
                    // RelatedField must be a BusinessClassGroup or an LPL Text field that is a condition
```

Text Search Fields

```
// Fields to use in a Text Search for this Business Class
     (<FullFieldName> | <FieldGroupName>)…              // will text search index persistent fields only
         [is a facet]
         [(index words | index phrase)]                                  // default is index words
         [(index data translations | index base data only)]              // default is index translations
```

Cube Relations

```
     <CubeRelationName>
         (cell | matrix) relation to <BusinessClass>
         [allow dimension reordering]                  // allows for dynamic optimization on retrieval
         [period view is <PeriodOperator>]
         [dynamic mapping is <RelatedValue>]
         [dynamic preload measures are <RelatedValue>]
         [Dimension Mapping]
             related.<DimensionField> ( = <RelatedValue> [and blank dimension]
                                      | where (<MeasureCondition>)
                                      )…
         [Preload Measures]
             <FullFieldName> [<PeriodOperator>]…     // must be a valid Measure on <BusinessClass>

     MeasureCondition ::= <Parens><MeasureConditionNode><Parens>
                          [<Conjunction> <Parens><MeasureConditionNode><Parens>…]

     MeasureConditionNode ::= <FullFieldName> [not] empty          // must be a valid Measure on <BusinessClass>
```

Relations

```
// By default, a relation is valid only if key fields are entered
     ( <RelationName> [is a[n] (<BusinessClass> | <KeyField>) set]
     | <KeyField> set
     )                          // When KeyField set is used, the RelationName is ' <KeyField>Set'. Alternatively a one-to-many relation
                                // based on a KeyField can be given a specific name by using '<RelationName> is a <KeyField> set'.
                                // A KeyField set is the preferred way to define a one-to-many relation. It is similar to a Relative
                                // Relation except that it automatically generates the relation to phrase below.
         [<SqlName>]
         [<ClassicName>]
         [<DefaultLabel>]
         [(one-to-one | one-to-many) relation (to <BusinessClass> | using <RelatedLink>)]
                                // If RelatedLink is a Relative Relation, it is a further definition based on the Relation given.
                                // This would typically be used only to further restrict a one-to-many relation. This phrase is required when the
                                // Relation is not a KeyField set, otherwise a KeyField set has the same rules as a Relative Relation.
         [valid when <Condition>]
         [<AsOfDate>]
         [required]
         [delete (restricted | cascades)]
         [(enable | disable) subset select on dependent relation]
         [disable filtering within db]
         [disable link cache]
         [include deleted record]       // Valid only with one-to-one where business class has a delete flag
                                        // When bus class is "store using" must implement DeleteFlag and support including deleted records
```

```
    [dynamic mapping is <RelatedValue>]
    [Field Mapping uses <Set>]      // Not required on a Relative Relation but can still be used to alter the sort order
        [related.<FullFieldName> <RelationOperator> <RelatedValue>…]
                        // Multiple dependent OTMs can be used in RelatedValue but only if all of the OTMs have a direct OTO
                        // relation back to the target of this Relation
    [Instance Selection]
        [include [only] deleted records]
        where <Condition>
                        // valid with one-to-many only. This expression can use the OTM object's fields and the current object's fields.

Set ::= ( <SetName>
        | [<BusinessClass>] part of key
        | [<KeyField>] (symbolic | level) key
        | bod key
        | update stamp key
        )
                        // BusinessClass must be one of the 'part of' business classes and KeyField must be one of the symbolic key fields
                        // 'level key' is valid only on a key field that implements the Array Hierarchy pattern
                        // 'bod key' is valid only when this business class implements BODId
                        // 'update stamp key' is valid when the update stamp exists. It is a virtual index except when the business class is a cube
                        // and only has the update stamp time stamp field on it.

RelationOperator ::= ( =
                     | >=
                     | >
                     | <=
                     | <
                     | as of
                     )
                     // non = operators are valid on last key only
                     // 'as of' operator is valid on a date only

<RelationName>       // A conditional relation must return the same underlying business object with the same cardinality (oto or otm)
    [<SqlName>]
    [<ClassicName>]
    <ConditionalRelationControl>

ConditionalRelationControl ::=
    if <Condition>
        (<RelationName> | <ConditionalRelationControl>)…
    [else]
        (<RelatedValue> | <ConditionalRelationControl>)…
```

Form Invokes

```
<FormInvokeName>
    (<FormInvokeRule> | <ControlRule>)…

FormInvokeRule ::=
    invoke <InvocationTarget>
        [fill in [blank] fields from <RelatedLink>]
            [except invoked.(<FullFieldName> | <FieldGroupName>)…]
                        // <RelatedLink> cannot be the same as the invoke's <RelatedLink>
        [invoked.<FullFieldName> <AssignmentOp> <RelatedValue>…]
        [<Rule>…]
                        // the way to address fields in the <RelatedLink> that is being addressed here is to use the
                        // invoked keyword
```

Matrix Forms

```
<MatrixForm>
    rows are <RelatedLink>
    ( column is <BusinessClass>
    | columns are <RelatedLink>
    )
    cell is <BusinessClass>
        [completion message is <Message>]
        [( create all instances    // if neither of these are specified then the matrix is view only
         | create instance
            when <Condition>
         )
        ]
```

Sets

```
<SetName>                          // NoKeyChange is no longer supported. All indexes can have keys changed.
    [<SqlName>]
    [<ClassicName>]
    [primary]                      // One index only must be primary and cannot have duplicates, if no Ontology section
    [override primary]             // Overrides the Set generated from the Ontology definition
    [( no duplicates               // default
     | duplicates )]
    [bypass no duplicates validation]
    [( indexed                     // 'indexed' forces the creation of a physical SQL index for faster retrieval
     | not indexed )]              // 'not indexed' prevents a physical SQL index from being created – it is always virtual
                                   // If neither of these are specified then the build process will dynamically create
                                   // a physical index based on general implementation parameters. Indexes that have
                                   // a related field in them cannot be physically indexed by the database.
    [primary when importing]       // use this index as the primary index when importing data
    [filter instance selection]    // filter instance selection rather than building a subset index
    Sort Order
        <FullFieldName> [descending]
        …
    [Instance Selection]
        where  ( <Parens><RelatedCondition><Parens>
                                   // for non-filtered selections, you must define a condition with only primary fields
               | <Condition>
               )
```

Rules

General Form for Rules

Contexts in which Rules can exist

```
Field Rules                                  SubType <ConditionName> Field Rules
    <FieldName>                                  <FieldName>
        <Rule>                                       <Rule>

Attach Rules                                 Dynamic Creation Rules
    <Rule>                                       <Rule>

State Rules                                  Attach Rules
    <Rule>                                       <Rule>
    <FieldName>                                  <FieldName>
        <Rule>                                       <Rule>
```

```
AssignmentOp ::=  ( =
                  | +=
                  | -=
                  | *=
                  | /=
                  | ^=          // exponentiation
                  | %=          // is modulus operator
                  )

Rule ::=  ( <ControlRule>
          | <GeneralRule>       // valid in all contexts
          | <FieldRule>         // valid in Field context only
          | <FieldCreateRule>   // valid in Field context with Create action
          | <GroupRule>         // valid in a GroupField only
          | <ArrayRule>         // valid in an ArrayField only
          | <ActionRule>        // valid in Action context only
          | <FormInvokeRule>    // valid in the Form Invokes section only
          )

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[<Parens><LPLConstructName><Parens>] ( [<Distinct> in] <otm RelatedLink>
                                                 | <array RelatedField>
                                                 | <iteration FullFieldName>
                                                 )
        [while <Condition>]
        <Rule>…
        [each[<Parens><LPLConstructName><Parens>].<RelatedField>]
                    // In 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 each1' 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.OccuringField) 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'.
```

Example of 'end for each'

```
for each X
    rules
    if (condition)
        end for each
    rules
    if (other condition)
        end for each
    rules
```

```
    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'.

GeneralRule ::=         // valid in all contexts
    confirmation required
        [<Message>]

    constraint <Condition>
        <Message>       // Message required on constraint rule

    <RelatedField> <AssignmentOp> <RelatedValue>                    // Assignment rule
                        // <RelatedField> valid only in scope of Business Object (not field rules)
                        // useful in a Field Context Rule

    increment <RelatedField> [by <RelatedValue>]
        [when <Condition>]

    decrement <RelatedField> [by <RelatedValue>]
        [when <Condition>]

    initialize [(<FullFieldName> | <FieldGroupName>)]

    (display | log) <Message>       // display message on console or write to standard log
                                    // when running on a production system 'display' is a noop

    add action tag <ActionTag>      // add an ActionTag to the execution path

    include <RuleBlockName>         // copy named block of rules in-line

    round <FullFieldName> [(up | down)] to nearest [exact] <RelatedValue>
                                    // if 'exact' is not specified then this rounds to the nearest multiple of
                                    // 'RelatedValue'otherwise it rounds 'exactly' to the specified RelatedValue
                                    // Thus if 'exact .95' is specified then the number will always end in '.95'
                                    // If non-exact '.01' is specifiied then the number will always end in a multiple
                                    // of '.01' – which in this case is simply rounding the number to the hundredths place

    clear in-memory cache [for <BusinessClass>]
                                    // this will clear the in-memory cache of all BusinessClass instances or a specific set of
                                    // BusinessClass instances if a BusinessClass is specified
```

```
    synchronize on <RelatedLink>
                                    // this will cause all the threads where RelatedLink points to the same record to single-
                                    // thread from this point until the end of the transaction

FieldRule ::=                       // valid in Field context only
    [(force | dynamic)] default [<FullFieldName>] to <RelatedValue>
                                    // a force default rule defaults the field whether it is entered or not
                                    // a dynamic default rule leaves the field blank and dynamically defaults it when it is referenced
                                    // a standard default fills in the field if it is blank
        [( default as a group       // default
         | default individual fields
         )
        ]

    required
        [<Message>]                 // defaults to 'Field is required'

    cannot be entered
        [<Message>]                 // defaults to 'Field cannot be entered'

    cannot be changed
        [<Message>]                 // defaults to 'Field cannot be changed'

    must be numeric
        [<Message>]                 // defaults to 'Field must be numeric'

    initial value is <RelatedValue>             // valid on Field Rules and Parameter Rules only
        [when <SimpleCondition>]

    field size is <RelatedField>

FieldCreateRule ::=   // valid in Field context with Create action when FullFieldName not used otherwise valid general rule
    autosequence [<FullFieldName>] [when blank] [using (<RelatedField> | <SetName>)]
        [manual range is <RelatedField>]        // must be a group field that implements FieldRange
        [minimize contention [and gaps]]
                    // if no field specified find last instance in context and increment by 1 – field must be either symbolic or relative key
                    // if SetName specified then find last instance using SetName – SetName must contain this field
                    // if RelatedField specified then increment RelatedField by 1 and use that value
                    // minimize contention is not valid with a RelatedField –this option can result in gaps in the numerical sequence
                    // typically 'minimize contention' will retrieve a block of numbers, when gaps are minimized as well, it retrieves one at a time
                    // 'when blank' is not valid with 'minimize contention'

GroupRule ::=                               // valid in a GroupField only
    always default  ( individual fields     // prevents group default rule from being overridden
                    | as a group
                    )

    mutually exclusive
        [<Message>]

ArrayRule ::=         // valid in an ArrayField only
    array size is <RelatedField>
    edit increasing sequence
    make increasing sequence
    edit decreasing sequence
    make decreasing sequence
    edit contiguous
    make contiguous
    no duplicates [on <FullFieldName> [, <FullFieldName>…]]
                    // where Field is a MemberField within an occurring GroupField
```

```
ActionRule ::=          // valid in Action context or RulesField only
    make transition to <FullStateName>

    commit transaction
                        // transactions are automatic—this forces a commit in the middle of a transaction and starts a new transaction
                        // this operation can cause data integrity problems unless the application is carefully constructed to manage this.
                        // on a SetAction this rule schedules a commit to take place at the next possible boundary—no data integrity problems
                        // will occur using this in a SetAction

    end set action instance loop
                        // valid in instance rules in a SetAction only—this ends the instance action loop but will continue to run Set Rules

    build text search field <FullFieldName>     // strings the given fields into <FullFieldName>
        Fields
            (<RelatedValue> | [<RelatedLink>.]<FieldGroupName>)…

    initiate <RelatedValue> workflow            // RelatedValue must be a WorkflowDefinitionCode
        [resume on error]                       // This will catch all errors thrown and then run the specified rules, if any
            [<ResumeOnErrorRule>…]
        [Variables]
            [include [(persistent | auditable | user)] fields from <RelatedLink>]…
                [include [(persistent | auditable | user)] fields from <RelatedLink>]…
                    [include [(persistent | auditable | user)] fields from <RelatedLink>]…
            [<RelatedField>…]
                [variable name is <LPLConstructName>]       // default is the last field name in the RelatedField
        [assign message id to <FullFieldName>]              // must be of type IONOutboxQueue

    cancel <RelatedValue> workflow              // RelatedValue must be an WorkflowDefinitionCode
        message id is <FullFieldName>           // must be of type IONOutboxQueue

    ( initiate <PFlowServiceName> process
    | trigger <RelatedValue> PA service)        // RelatedValue must be a literal or PfiServiceDefinition key field
        [resume on error]                       // This will catch all errors thrown and then run the specified rules, if any
            [<ResumeOnErrorRule>…]
        [title is <Message>]
        [Criteria]
            <RelatedField>…
        [category filter is <RelatedField>]
            [variable name is <RelatedValue>]               // default is the last field name in the RelatedField
        [Variables]
            [include [(persistent | auditable | user)] fields from <RelatedLink>]…
                [include [(persistent | auditable | user)] fields from <RelatedLink>]…
                    [include [(persistent | auditable | user)] fields from <RelatedLink>]…
            [<RelatedField>…]
                [variable name is <LPLConstructName>]       // default is the last field name in the RelatedField
        [URLs]
            (<HttpURL> | <LinkBack>)…           // list of urls to show for a participant of the process
                                                // Maps to the 'Folders' concept in ProcessFlow

    ( cancel <PFlowServiceName> process
    | cancel <RelatedValue> PA service)         // RelatedValue must be a literal or PfiServiceDefinition key field

    invoke <InvocationTarget> [in  ( foreground
                                   | background [on <RelatedField>]
                                   | background group[<Parens>(<RelatedValue>)<Parens>]
                                   )
                               ]
                        // <CreateAction> must be a non-parameter Create Action
```

```
                    // invoking a Create action using [this instance] can be used only in the context of a Create or Import action
                    // RelatedField must be of type Date or TimeStamp
    [<InvokeRule>…]
    [<Rule>…]
    [<ControlRule>]
        (<InvokeRule> | <Rule>)…

invoke modify update stamp <RelatedLink>

invoke method <NativeMethod>

invoke script <RelatedValue>                // RelatedValue is a UserScript key field value

invoke derived field <RelatedValue>         // RelatedValue must be a derived field

generate document <RelatedValue>            // RelatedValue must be '<NavigationName> as pdf…'
    set using action <ActionName>           // Action must have a parameter of type BinaryDocument
                    // This will generate the document in the background and use the Action specified to set it on the calling instance

dbimport <RelatedField> into <BusinessClass>  // RelatedField must be of type CSVText
                    // This has the same behavior as the dbimport command except that it can also take the
                    // standard output format of a csv-based ReplicationSet (the ability to create, update, and delete)

send ion bod
    bod is <RelatedValue>                                   // must be an XMLDocument
    bod type is <RelatedValue>                              // <verb>.<noun>
    [assign message id to <FullFieldName>]                  // must be of type IONOutboxQueue
    [(expect | no) acknowledgement]                         // default is "expect acknowledgement"
    [accounting entity is <RelatedValue>]
    [location is <RelatedValue>]
    [document id is <RelatedValue>]
    [variation id is <RelatedValue>]
    [revision id is <RelatedValue>]
    [source is <RelatedValue>]
    [instance count is <RelatedValue>]
    [source is <RelatedValue>]
    [custom property is <RelatedValue>]                     // must be an IONCustomerProperty

send notification
    to <RelatedValue>           // must be an Actor or ActorGroup
    description is <Message>
    [priority is (very low | low | medium | high | very high)] // default is medium
    [detail is <Message>]
    [<LinkBack>]
    [navigation is <NavigationName>]        // @deprecated 20180809

send email
    [to <RelatedValue>…]        // must enter at least one to, cc, or bcc; email addresses cannot include spaces
    [cc <RelatedValue>…]        // can have multiple to, cc, bcc address lines, each having a single address, or can enter multiple
    [bcc <RelatedValue>…]       // addresses on one line using a valid delimiter, such as a comma or semicolon (not a space)
    from <RelatedValue>…        // repeat is for <RelatedValue> only
    [ignore invalid addresses]
    [subject <Message>]
    [<Attachment>…]
    [Attachments]
        (<Attachment> | <ControlRule>)…
    [<Appointment>…]
    [Appointments]
        (<Appointment> | <ControlRule>)…
```

```
        Contents
            (<Message> | <ControlRule where <Rule> is replaced by <Message>>)…

    link <RelatedValue> to this agent          // RelatedValue must be an Actor

    register new actor
        login name is <RelatedValue>
        password is <RelatedValue>
        webapp is <WebAppName>
        [registration key is <Message>]
        [role is <RelatedValue>…]
        [person name is <RelatedValue>]
        [contact info is <RelatedValue>]

    update actor
        [person name is <RelatedValue>]
        [contact info is <RelatedValue>]

PFlowService ::= (<PFlowServiceName> | <RelatedValue>)      // RelatedValue must be a PfiServiceDefinition key field

Attachment ::=
    attachment <RelatedField>
        [name is <RelatedValue>]
        [mime type is <RelatedValue>]

Appointment ::=
    appointment <RelatedValue>
        [send as (meeting | [appointment] message | [appointment] attachment)] // message is default
        [name is <RelatedValue>]
        [location is <RelatedValue>]
        [cancel]
        date [range] is <RelatedValue> [(to <RelatedValue> |
                                         for <RelatedValue> days | hours | minutes)]
        [reminder <RelatedValue> days | hours | minutes] before]
                    // Must supply a globally unique appointment RelatedValue
                    // Name is ignored for message types; becomes attachment name for others.
                    // 'date is <RelatedValue>' is required for all cases. All cases also require 'to' or 'for'.
                    // Meeting is sent as an email attachment prepopulated with attendees and is only sent to the originator who can send
                    //     invites from their calendar upon opening the attachment. Must be an attachment due to calendar client limitations.
                    // Message is embedded in the email message and is sent to each attendee. This is the default if nothing is specified.
                    // If there is more than one appointment that is a message, only the first is embedded. The rest become attachments
                    //     due to calendar client limitations.
                    // Attachment is sent as an email attachment to each attendee and must be opened to get into the calendar.

ResumeOnErrorRule ::= (<Rule> | cancel resume | <FullFieldName> = error message)

InvokeRule ::=          // valid under invoke rule only
    run outside of action background group
                    // Every top level action creates an implicit aciton background group that, by default, all invoked actions run in. This action
                    // background group is used to define what it means for the top level action to be completed. This syntax allows for
                    // an action to be executed in the background that is not tied to the top level action such that the top level action will be
                    // considered to be completed even if this action is still running. All sub-actions of a top level action dynamically become
                    // a part of the action background group of the top level action. This is not true for named background groups–they have
                    // to be explicitly joined when invoked.

    run after  ( <FullFieldName>   // must be an AsyncActionRequest KeyField / UniqueID or a named background group id
               | current action background group
               | action background group <FullFieldName>     // must be an action background group id
               | background group<Parens><RelatedValue><Parens>
               )
```

```
    on error          // valid only when action is run in background
        invoke <InvocationTarget>
            [<Rule>…]
            [<ControlRule>]
                (<InvokeRule> | <Rule>)…

    assign async action request id to <FullFieldName>        // valid only when run in background
                      // FullFieldName can be an AsyncActionRequest KeyField or a generic UniqueID

    assign async [action] background group id to <FullFieldName>  // valid only when run in background
                      // FullFieldName must be a generic UniqueID

    assign result to (<KeyField> | <ViewField> | <BusinessObjectReference>)
                      // This defaults when the InvocationTarget is a KeyField
                      // Not valid with 'create instance'
                      // This must be a BusinessObjectReference if the InvocationTarget is an update type of
                      // Lawson Classic Transaction.
                      // This can be used only on a Lawson Classic Transaction if it has a business class designation.

    assign message group id to <LawsonClassicMessageGroup KeyField>

    resume on error   // This will catch all errors thrown on this action invocation and then run the specified rules, if any
        [<ResumeOnErrorRule>…]

    fill in [blank] user fields from (parameters | <RelatedLink>)

    fill in [blank] audit fields from <RelatedLink>
                      // <RelatedLink> cannot be the same as the invoke's <RelatedLink> and must be either 'this instance' or
                      // 'each(<LPLConstructName>)' where the target of the each is an audit log entry

    fill in [blank] fields from <RelatedLink>
        [except <InvokeRuleKeyword>.(<FullFieldName> | <FieldGroupName>)…]
                      // <RelatedLink> cannot be the same as the invoke's <RelatedLink>

    <InvokeRuleKeyword>.<FullFieldName> <AssignmentOp> <RelatedValue>…
                      // the way to address fields in the <RelatedLink> that is being addressed here is to use the invoked keyword

    initialize <InvokeRuleKeyword>.<FullFieldName>…

    <FullFieldName> <AssignmentOp> result.<FullFieldName>…
                      // A LawsonClassicTransaction is capable of returning result fields. The result fields are accessed using the 'result' keyword.

InvokeRuleKeyword ::=  ( invoked
                       | datasource       // datasource keyword valid only when using the 'implement xxx DataSource' pattern
                       )

InvocationAction ::=   ( [<FullStateName>].<ActionName>
                       | audit create
                       | audit update           // must have a RelatedLink that is an audit log entry
                       | change effective date // must have a RelatedLink that is an audit log entry
                       | insert history
                       | delete history
                       | change create date
                       | purge audit log entries
                       )

InvocationTarget ::= <InvocationAction>[ ( <RelatedLink>
                                         | <BusinessClass>  // must be a Set Action (construct also option in
                                                            // <RelatedLink>, but restated here for emphasis
```

```
                                                    // and Set Action exception
                                    | <BusinessTask>    // any action on a BusinessTask
                                    | <LawsonClassicTransaction>
                                    | <M3Interface>
                                    | <WebserviceInterface>
                                    | <StaticJavaPD>
                                    )
                                ]

FieldRules ::=
     <FullFieldName>    // Field name can have template operator in it '<A>'; if it does then rules can have matching template operator
          <Rule>…
```

Field Rules

```
[(merge | replace) specifications]              // valid only for lhf LPL, merge is default
<FieldRules>…
```

Translation Field Rules

```
[(merge | replace) specifications]              // valid only for lhf LPL, merge is default
<FieldRules>…          // Rules that get 'fired' when this field's translation is changed (this includes Create and Delete)
```

SubType <RelatedCondition> Field Rules

```
[(merge | replace) specifications]              // valid only for lhf LPL, merge is default
<FieldRules>…
```

Commit Rules

```
<Rule>…                // Rules that get 'fired' just before the final 'commit' point of any transaction.
                       // Commit Rules do not get fired when a RequestAction is initiated; they get fired when it is completed.
                       // Commit Rules do not get fired when a future effective-dated transaction is initiated; they get fired when
                       // the transaction becomes effective.
```

Audit Entry Rules

```
<Rule>…                // Rules that get 'fired' just before an audit log entry is created. Multiple audit log entries can be created in a single
                       // transaction if the effective date has been changed in the logic of the transaction.
                       // Audit Entry Rules do not get fired when a RequestAction is initiated; they get fired when it is completed.
                       // Audit Entry Rules DO get fired when a future effective-dated transaction is initiated; they DO NOT get fired
                       // when the transaction becomes effective.
```

Apply Pending Effective Rules

```
<Rule>…                // Rules that get 'fired' when a pending effective-dated transaction is applied (when it becomes 'effective')
```

Create Rules

<Rule>... // Rules that get 'fired' when any Create type action is executed

Create Exit Rules

<Rule>... // Rules that get 'fired' after any Create type action is executed

Update Action Rules

<Rule>... // Rules that get 'fired' when any Update type action is executed

Delete Rules

<Rule>... // Rules that get 'fired' when any Delete type action is executed

Translation Rules

<Rule>... // Rules that get 'fired' when any translatable field's translation is changed (this includes Create and Delete)

Action Exit Rules

<Rule>... // Rules that get 'fired' on the exit of all actions (except Set Actions) defined on this business class

Attach Rules

<Rule>... // Rules that get 'fired' when another business object is seeking to 'attach' itself to this object
// Only rules that are valid in all contexts. Any constraint rule means that attachment of calling object is not allowed.
// No <FieldRules>

Parent Attach Rules

<Rule>... // Valid on Parent Hierarchy only. Rules that get 'fired' when a child attempts to attach to a parent.
// Only rules that are valid in all contexts. Any constraint rule means that attachment of calling object is not allowed.
// Rules within Parent Attach Rules can use the 'child' related link to access the attaching child business class.
// No <FieldRules>

Dynamic Creation Rules

<Rule>... // Rules that get 'fired' when a Dynamic Creation is about to occur
// Any constraint rule means that Dynamic Creation does not occur and an exception does occur.
// Both Attach Rules and Dynamic Creation Rules can address contextual information about the business class
// that is causing these rules to fire through the special variable datacontext.

Rule Blocks

```
<RuleBlockName>…        // Blocks of rules that can be included (copied) in multiple places in the LPL code
    <Rule>…             // The block of rules is copied into place during compilation when an 'include <RuleBlockName>' rule is used.
                        // Rule Blocks can contain replacement variables; these are LPLConstructNames bounded by curly braces.
                        // If a Rule Block contains replacement variables, all of those variables must be replaced when including the Rule Block
```

Examples

```
Rule Block
    AssignGLVariables
        FinanceDimension1 = {GLDimension1}

Action Rules
    include AssignGLVariables
        replace GLDimension1 with APInvoice.FinanceDimension1

Attach Rules
    if (parentcontext.istransaction)
        UsedByATransaction = true

Parent Attach Rules
    if (child.Supplier != Supplier)
        constraint (child.Supplier = any Contract ancestors.Supplier)
            "ParentContractMustBeWithinTheSupplierHierarchy"
```

States and Actions

// State definitions are optional. If States are defined, Actions can be defined within States or outside of the State definitions (meaning for all States). If no
// Actions are defined, the actions Create, Update, and Delete are automatically generated. If any actions are defined, one of them must be a Create type
// of action (it is ok to not be able to update or delete a business object but if it can never be created, there isn't much point in defining it in the first place).

StateCycles

```
<StateCycleDefinition>

StateCycleDefinition ::=
    <StateCycleName> is a StateCycle
        [state field is <persistent FieldName>]          // default is <StateCycleName> + State
        [initial state is <StateName>]                   // default is first state definition
        <StateDefinition>…

StateDefinition ::=
    <StateName> is a State
        Entrance Rules
            <Rule>…
        Exit Rules
            <Rule>…
        Field Rules
            [(merge | replace) specifications]           // valid only for lhf LPL, merge is default
            <FieldRules>…
```

```
SubType <ConditionName> Field Rules
    [(merge | replace) specifications]              // valid only for lhf LPL, merge is default
    <FieldRules>…
[<EventHandler>…]
[<ActionDefinition>…]
[<StateCycleDefinition>…]
```

// A StateName must be unique within another State. StateCycles do NOT participate in the name scoping of States. Therefore, if there is a state 'Open' that has two
// StateCycles ('ApprovalCycle' and 'NotificationCycle') the StateNames within those two StateCycles must be unique. This allows a State to be fully identified
// via its upper StateName. In this example if ApprovalCycle has a State 'NeedsApproval' then it can be addressed using 'Open.NeedsApproval'. A StateName
// address need not be fully qualified as long as it can be unambiguously resolved.

Actions

```
(<ActionDefinition> | <EventHandler>)…

ActionDefinition ::= // When you specify any actions in the action block, you must also SPECIFICALLY include the following action if you
                     // want them available:Create is an Action, Update is an Action, Delete is an Action
    <ActionName> is a[n] [<ActionType>] [Request]Action
                                    // A blank ActionType means Instance Action unless ActionName is the same as ActionType.
                                    // If ActionName == ActionType, ActionType = ActionName
        [<DefaultLabel>]
        [privileged]
        [action tag is <ActionTag>]…
                                    // When an audit log entry is created all the action tags of all the actions in the execution path
                                    // are put on the audit log entry. This is useful for determining a reason for why an audit log entry
                                    // exists.
        [completion message is <Message>]
        [no records message is <Message>]       // valid on Set actions only
        [workflow event title is <Message>]     // valid on Request actions only
        [valid when <Condition>]
            [<Message>]
        [restricted]                                        // can be invoked only; cannot be executed via UI
        [(restrict | enable) action on lists]               // overrides business class default
        [(refresh and | lazily) lock this instance]         // valid on Instance, Update, Delete, and Purge actions only
        [(enable | disable) ImmutableContextCache]          // disabled by default unless enabled in Product Line Definition
        [bod type is <Literal>]                             // valid on Import actions only '<verb>.<noun>'
        [bypass field rules]                                // valid on Create and Update actions only
        [bypass relational integrity rules]                 // valid on Delete and Purge actions only
        [use array insert]                                  // valid on Create actions only
        [manual update]
        [disable checkpoint]                    // Set Actions only—disables restart logic; implies allow unlimited concurrency
        [subject is <Subject>]
        [effective date required]
        [reason code required]
        [action comment required]
        [request action process is <PFlowServiceName>]
        [request action linkback webapp is <WebAppName>]
                                    // The main <LinkBack> syntax declares a linkback with its associated parameters (in parens).
                                    // This statement modifies an implicit 'linkback webapp is' statement.
        [disable RetroactiveEffectiveDating]
        [disable resume on error]
        [disable multiple instance selection] // Do not allow multiple records to be selected on a list for this action
        [confirmation required]               // Causes the user to confirm they want to execute this action
            [<Message>]
        [run in (foreground | background)]    // Overrides default run mode – valid on Instance and Set Action only
            [[initial] schedule concurrency is ( FullConcurrency    // valid only on run in background
                                               | NoConcurrency
```

```
                                          | NoActionGroupConcurrency)]
    [[initial] misfire strategy is ( DoNothing          // valid only on run in background
                                   | RunOnce            // default is DoNothing
                                   | RunAllProcessesScheduled)]
[allow anonymous access]                 // Allows the 'anonymous' user to execute this action
[allow user fields [from <BusinessClass>] as parameters]…
             // Allows user fields to be placed on the ActionForm during configuration.
             // User fields from multiple business classes may be used as parameters
             // These user field parameters can be used in the invoke rule 'fill in user fields from parameters'
[synchronized [on <Message>]]            // enforced only on actions run in the background
             // If using a translatable field in the message, must force the locale to be blank
[allow unlimited concurrency]            // valid on Set actions only—implicit with 'disable checkpoint'

Queue Mapping Fields
    <RelatedField>…                      // can have up to two Queue Mapping Fields

Set Is                                   // valid on Set actions only
    <RelatedValue>…

Parameters                               // valid on Create, Instance, and Set actions
    <FieldName> [<DataDefinition>]…
        [<States>]
        [<DefaultLabel>]
        [holds pii]
        [scannable]
        [as of <RelatedValue>]
        [within <RelatedValue>]
            [<Message>]
        [[exact] version is <RelatedValue>]
        [precision is <RelatedValue>]
            [round to precision]
        [old value is <RelatedValue>]
        [<TextVariables>]

    <FieldName> is a[n] <BusinessClass> view

Parameter Rules
    <parameter FullFieldName>
        <Rule>…

Local Fields
    // same definition as on Business Class

Results                                  // valid on Set actions only
    (<FieldName> | <BusinessClass> view)…

Field Rules                              // valid on Create and Update actions only
    [(merge | replace) specifications]           // valid only for lhf LPL, merge is default
    <FieldRules>…

SubType <ConditionName> Field Rules      // valid on Create and Update actions only
    [(merge | replace) specifications]           // valid only for lhf LPL, merge is default
    <FieldRules>…

Accumulators
    <FieldName> [<DataDefinition>]

Instance Selection [<AsOfOperator>]      // valid on Set actions only
    [include [only] deleted records]
```

```
        where <Condition>

    Sort Order [is (<SetName> | primary)]  // valid on Set Actions only
        <RelatedField>…
            [is transaction boundary]

    Action Rules                    // Rules for entire set if a Set Action – otherwise just the rules for this instance
        <Rule>…                     // valid on a non Set Action only - constraint type rules as well as action rules
                                    // can be placed here – constraint rules constrain whether the action takes place or not
        Empty Set Rules             // valid on Set actions only
            <Rule>…

        Set Rules                   // valid on Set actions only
            Entrance Rules
                <Rule>…

            Exit Rules
                <Rule>…

        <sort FullFieldName> Set Rules…     // valid on Set actions only
            Entrance Rules
                <Rule>…

            Exit Rules
                <Rule>…

        Instance Rules              // valid on Set actions only
            <Rule>…

    Entrance Rules                  // Rules to be executed prior to the Action Rules – this is primarily useful for delete actions
        <Rule>…                     // because in a delete action the record is deleted before the Action Rules fire

    Exit Rules                      // Rules to be executed upon exit of the Action – this is primarily useful for create actions
        <Rule>…                     // because in a create action the record is created after the Action Rules fire

    InitiateRequest Rules           // valid on RequestAction only
        <Rule>…                     // Rules are executed when the action request is initiated

    UpdateRequest Rules             // valid on RequestAction only
        <Rule>…                     // Rules are executed when the action request is updated

    CancelRequest Rules             // valid on RequestAction only
        <Rule>…                     // Rules are executed when the action request is canceled (rejected or withdrawn)

    Rule Blocks
        <RuleBlockName>…
            <Rule>…

ActionType ::= ( Create             // Creates new records and fires all field rules
               | Update             // Updates record and fires all field rules
               | Delete             // Virtual delete. Marks record as 'deleted'. Does not physically remove record from database.
               | Purge              // Physical delete. Removes record from database.
               | Preview            // Similar to an update but it does not fire field rules and does not actually update record
               | Instance           // Does not run field rules but does update record
               | Set                // Runs rules on a set of records
               | Import             // Implicitly runs in the context of a new instance but does not implicitly save that instance
               )
```

```
EventHandler ::=
     on (<FullStateName>.<ActionName>
        | entrance to <FullStateName>
        | exit of <FullStateName>
        )
          Action Rules
              <Rule>

ActionOverride ::=                    // valid on configuration only
     [<FullStateName>.]<ActionName> is an ActionOverride
          [<DefaultLabel>]
          [effective date required]
          [reason code required]
          [action comment required]
```

Example of State syntax

```
LifeCycle is a StateCycle

    state field is LifeCycleState  // default is <StateCycleName> + State

    Unreleased is a State
        Create is an Action
        Update is an Action
        Delete is an Action
        Release is an Action
            Action Rules
                make transition to Released
                invoke Release Unreleased Polines

    Released is a State
        Cancel is an Action
            Action Rules
                constraint (NoActivity)
                    "Cannot cancel if Activity exists"

                    make transition to Cancelled

        Update is an Action

        Approval is a StateCycle
            initial state is NeedsApproval

            Approved is a State
                Update is an Action
                    Action Rules
                        if (any audited field changed)
                            make transition to NeedsApproval

            NeedsApproval is a State
                Update is an Action
                    Action Rules

                Approve is an Action
                    Action Rules
                        make transition to Approved

        Communication is a StateCycle

            NotIssuable is a State
                On NeedsApproval.Approve
                    Action Rules
                        make transition to Issuable

            Issuable is a State
                Issue is an Action
                    Action Rules
                        make transition to Issued

            Issued is a State
                Update is an Action
                    Action Rules
                        if (any audited field changed)
                            make transition to Revised

            Revised is a State
                On NeedsApproval.Approve
                    Action Rules
                        make transition to Issuable

    Cancelled is a State
        Close is an Instance Action

    Closed is a State
        Delete is an Action

        Unrelease is an Instance Action
            Action Rules
                make transition to Unreleased
```

Rule Blocks Definition File

// Rule Blocks files have an extension of .ruleblocks

RuleBlocks Structure ::=

```
<RuleBlocks> is a RuleBlocks
    Rule Blocks
```

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 Backs]
        [<DrillBackDefinition>…]
    [Drill List]                          // When this section is present it overrides automatic drill propagation
        [(<DrillDefinition> | primary audit list | data translation list)…]
    [<DrillListDefinition>…]
    [<NavigationDefinition>…]
    [<ContextMessageDefinition>…]
    [Context Message Invocations]
        [<ContextMessageInvocation>…]
    [<ListDefinition>…]
    [<CardViewDefinition>…]
    [<InstanceCountChartDefinition>…]
    [<FormDefinition>…]
    [<ActionFormDefinition>…]
    [<CompositeFormDefinition>…]
    [<MatrixFormDefinition>…]
    [<SearchFormDefinition>…]
    [<SummaryFormDefinition>…]
```

Base Definitions

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

```
ActionIcon ::=  ( add
                | apply
                | approve
                | award
                | calculate
                | chart
                | claim
                | copy
                | create
                | delete
                | edit
                | export
                | filled star
                | filter
                | finalize
                | open
                | outlined star
                | pdf
                | preview
                | print
                | process
                | reject
                | release
                | request
                | save
                | search
                | send
                | submit
                | transfer
                | undo
                | validate
                )

AggregationDrillLink ::= row[<Parens><SetName><Parens>][.<RelatedLink>]

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>)
    | [row] form invoke is <FormInvokeName>
    | [row] navigation is <NavigationName>          // navigation and row navigation both operate off the row
    | [row] link is <UILink>
    )
        [action icon is (<ActionIcon> | <Icon> | hidden)]
                                    // Icon must be a SOHO icon: https://design.infor.com/product/identity/icons
        [label is <Message>]        // Required when using form invoke is...
        [mouse over text is <Message>]
        [hide when invalid]
        [valid when <SimpleCondition>]

CaptchaControl ::=
    captcha control
```

```
        [(valid | visible) when <SimpleCondition>]
        [(left align | center | right align)]      // left align is default

CheckControl ::=
    check control
        [(valid | visible) when <SimpleCondition>]
        [checked state is <FieldName>]
        [check action is [<StateName>.]<ActionName>]…          // optional for printing purposes
            [when <SimpleCondition>]                           // must have a condition if defining multiple targets
        [uncheck action is [<StateName>.]<ActionName>]…
            [when <SimpleCondition>]                           // must have a condition if defining multiple targets
        [display as (box | switch | star)]                     // default is box
        [<Label>]                                              // default is no label
        [mouse over text is <Message>]
        [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]                          // make check control line up in label column
        [indent]
        [hidden]
        [when value changed]
            ( validate (this field | <RelatedField>)           // RelatedField must be a mutable field on a form or list
            | <RelatedField> = <RelatedValue>                  // RelatedField must be a mutable field on a form or list
            | refresh <RelatedField>                           // RelatedField must be on a form or list
            | refresh dependent panels
            )…
                [when <SimpleCondition>]

Color ::= ( black
          | blue
          | brown
          | gray
          | green
          | jade          // deprecated
          | magenta       // deprecated
          | maroon        // deprecated
          | orange
          | purple
          | red
          | turquoise
          | yellow
          )

DisplayType ::= ( browser[(/v3 | /v4+)]
                | desktop
                | smartphone[(/v3 | /v4+)]
                | tablet[(/v3 | /v4+)]
                | v3
                | v4+
                )

FieldStyle ::= ( header(1|2|3|4|5)
               | (left align | center | right align)          // left align is default
               | (top align  | middle | bottom align)         // top align is default
               | bold
               | italics
               | color of <Color>
               )
```

```
Form ::= [independent] form is [<RelatedLink>.](<FormName> | primary | inline)

FormText ::=
    ( text of (<Message> [with <Icon>] | <Icon>)
        [<TextStyle>…]
        [link is <UILink>]
        [display as tag]
            [color of <Color> [when <SimpleCondition>]…]
        [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>]]
        [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> | random background)
                            // 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> | <AggregationDrillLink>).(<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 on an inline Form only; 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 on a CubeView only; overrides the CubeView Row Dimension
            <DimensionField>…                       // DimensionField is in context of the RelatedLink business class
        [filter list is <RelatedLink>.(<ListName> | primary)…]
            [valid when <SimpleCondition>]
            [group label is <RelatedValue>]
            filter <RelatedField> using <RelatedValue>…
                                // RelatedField is from the list business class and RelatedValue is from the filter list business class
```

```
            [implements DragAndDrop]
                action is [<StateName>.]<ActionName>
                    [invoked.<FullFieldName> = [drop target.]<RelatedValue>…]
        [helper list is <RelatedLink>.(<ListName> | primary)…]
            [valid when <SimpleCondition>]
            [title is <Message>]
            [search <FullFieldName> using <RelatedValue>…]
            [<Action>]
                [invoked.<FullFieldName> = (<RelatedValue> | parameter)…]
                        // 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.

ManualRepresentation ::=
    manual representation
        representation name is <Literal>
        [(label is (<RelatedField> | <Message> | number) | no label)]        // default is Field Label
            [<TextStyle>…]
        [align as label]                                    // make field line up in label column – valid only with no label
        [show up to <Literal> lines]
        [show up to <Literal> characters]
        Property Mapping
            <MessageProperty> = <RelatedValue>…
        [raise <AlertType> alert when <SimpleCondition>…]
            [mouse over text is <Message>]
            [link is <UILink>]

Navigation ::= navigation is <NavigationName>

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

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

StandardAction ::= ( open
                   | save [and (new | close)]
                   | inline create
                   | create within
                   | create menu
                   | export to (pdf | csv)
                   | search
                   | drill around
                   | refresh
                   | charts
                   | helper list
                   | next record
                   | previous record
                   | copy row
                   | audit compare
                   | sort
```

```
                    )

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

Template ::=
    template is <RelatedLink>
        [implements DynamicChildVisibility]
            child designator field is <FullFieldName>          // FullFieldName must be from the template business class
            parent match field is <FullFieldName>              // FullFieldName must be from the target business class
        [target] business class is <BusinessClass>// The parser accepts this line and the one that follows at this level of indent
            ( <Form>                                   //     or one less indent level – it also allows 'target' to be unspecified
                [<FormDefinition>]                     // valid on inline Form only
            | 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 target <BusinessClass> instances. <BusinessClass> must implement the TemplateDriven pattern. The
                        // <RelatedLink> business class must be the business class specified in the TemplateDriven pattern.
                        // The DynamicChildElementVisibility pattern allows children template elements to be defined by a non-blank
                        // FullFieldName on the template business class. The children template elements will only be visible if the value
                        // in the child designator field matches the value in the parent match field.

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 ::=    ( [<RelatedLink>.]<NavigationName>
              | [(embedded | external | dialogue)] <HttpURL>
                    [lazy url retrieval]
                    [mingle url is <Message>]
                        // 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 canvas) or on another tab
                        // 'dialogue' is valid only when in a browser and will mimic a dialogue
                        // mingle url will be used if mingle is installed; url must be of format '<logicalId>/<restofurl>'
              | registration
              | signin
              | signout     // page, menus, forms
              )

URL ::= url is <HttpURL>
```

Drills

```
DrillBackDefinition ::=
```

```
    <LinkBack>

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 | enable) 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 | data translation list)…
```

Navigations

```
NavigationDefinition ::=
    <NavigationName> is a Navigation
        [label is <Message>]
        [drill restricted]            // prevents this navigation from being put in the automated drill list
        [valid when <SimpleCondition>]
        [show as pdf [in (foreground | background)] [in (portrait | landscape)] [font offset
                      is <Number>]]
                        // By default, navigating to a PDF causes the PDF to be generated in the foreground and then displayed. This
                        // is useful when the PDF is just a more complex page and is not a long-running report.
                        // For long-running reports the PDF should be sent to the background.
        ( [<DisplayType>] <Page>
            [when <SimpleCondition>]
        | [<DisplayType>] <Form>
            [open action is <ActionName>]
            [(restrict | enable) action [<StateName>.]<ActionName>…]
            [<Form Definition>]
            [when <SimpleCondition>]
        | [<DisplayType>] <List>
            [position to <RelatedLink>]          // <RelatedLink> must be a record in the specified List
            [when <SimpleCondition>]
        | [<DisplayType>] <Template>
            [when <SimpleCondition>]
        | view <RelatedField>          // must be a BinaryDocument or a BusinessObjectReference. If RelatedField is a
            [when <SimpleCondition>]   // 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
                            // http://wiki.infor.com:8080/confluence/display/companyon/Infor10+Workspace+-+Standard+Context+Message+Definitions

MessageProperty ::= <Literal>

ContextMessageInvocation ::=            // valid on a BusinessClass definition only
    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
        [is report [for <WebAppName>[,<WebAppName>]…]
        [show <#> lines]
        [show result set size]
        [default Alpha filter operator is (contains | starts with | equals)]    // default is 'contains'
        [keyword search field is <RelatedField>]
            [label is <Message>]
            [(always | initially) display]
            [disable translation filtering]
        [(always | initially) display search]                   // valid for search form and filter
        [( search form is (<FormName> | inline)
         | search field is <FullFieldName>…)]
            [label is <Message>]                                // valid on 'search field is…' only
            [use as filter]
            [always display]
            [disable translation filtering]
            [Layout]                                            // valid on inline Form only
                <FormLayout>…
        [Required Search Parameters]
            <RelatedField>…
                [initial value is <RelatedValue>]
        [implements LongRunningList]
        [implements InlineCreate]
            [allow form create]
            [copy mutable fields only]                          // default copies all persistent fields in list
        [implements FixedSizeList]
            [show <#> lines]                                    // default is 10 lines
        [implements RepresentativeImageView]                    // business class must have a representative image defined
        [implements AuditCompare]                               // valid only for AuditList
            show changed values on right
        [implements TreeView]
```

```
    [show parent]
    [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 ColumnarView]
    [union with <BusinessClass>]…
    [show charts only]
[implements CubeView]
    [suppress blank rows]
    [suppress current period]              // by default the CubeView is in the context of the current period
    [period dimension is <DimensionField>] // default is the first period in the list of dimensions
    [show total node]
        [label is <Message>]
    [show <RelatedValue> levels]
    [Context]
        <FieldName>…
                // 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.
                // Values can also be directly entered in Filter or Search Form for use in other BusClass Context fields
    [Row Dimensions]                       // must have either Row Dimensions or Pivot Views defined
        <DimensionField>…
            [caption is (representative text | <Message>)]
            [is fixed]                     // fixed dimensions must start at the top and be contiguous

    [Pivot Views]
        <PivotViewName>…
            [label is <Message>]
            [is default]
            Row Dimensions
                <DimensionField>…
                    [caption is (representative text | <Message>)]
                    [is fixed]

    [Column Views]
        <ColumnViewName>…
            [label is <Message>]
            [is default]
            Display Fields
                <ListDisplayField>…

    [Period Views]                         // valid only if the cube has a period
        <PeriodViewName>…
            [label is <Message>]
            ( 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
            )

[implements DashBoardView]                 // valid only when using a CardView (without Display Fields) and Detail Sections
    [show (3 | 4) columns]                 // default is 3 columns
```

```
    [use 12 column grid]
    [print as dashboard]
        [compress columns]

[implements HorizontalScrolling]
    [freeze first [<RelatedValue>] column[s]]

[auto refresh using <RelatedField>]          // must be a TimeStamp field
    [raise <AlertType> alert]
        [mouse over text is <Message>]
        [link is <UILink>]
[is default select]
[is default TreeView select]
[show as simple list]
[show in descending order]
[show in relevance order]
[show totals only]
[suppress sub-totals]
[sub-total text is <Message>]
[grand total text is <Message>]

[display negative amounts using parens]
[display negative amounts in <Color>]
[display amounts in (thousands | millions)]
    [show in title]
[disable select in webui]                                  // deprecated
[(multiple | single | disable) select]                     // default is multiple

[no data message is <Message>]
    [description is <Message>]
    [icon is <Icon>]
    [<FormButton>]

    [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) // valid only on non-ColumnarView
     |sort field is <RelatedField>)]…      // valid only on ColumnarView
        [is default [descending]]
        label is <Message>

[responsive card view is <CardViewName>]   // valid only if "card view is…" is defined
    [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) // valid only on non-ColumnarView
     |sort field is <RelatedField>)]…      // valid only on ColumnarView
        [is default [descending]]
        label is <Message>

[Subtotal By]                              // default is all fields in sort order except last one
    <RelatedField>…                        // must be field in a sort order
        [page break]

[csv list is <ListName>]                   // allows for a different list to be used when exporting to csv or excel

// PDF generation / formatting only

[print list is <ListName>]                 // allows for a different list to be used when printing to pdf
[print form is <FormName>]                 // allows for a form to be used as the print layout
```

```
    [total form is <FormName>]
    [when printing detail sections]
[orientation is (portrait | landscape)]
[font offset is <Number>]
[show grid lines]                          // PDF formatting only
[suppress header]
[suppress footer]
[first page header is <FormName>]          // only prints on first page
[page header is <FormName>]                // will not print on first page
[page footer is <FormName>]

// End PDF generation / formatting only

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

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

[Summary Total Fields]
    <FormField>…

[Instance Selection]
    [include [only] deleted records]
    where <Condition>

[Additional Folder Items]                  // A record in a list can be copied to a folder. This allows for related
    <KeyField>…                            // records to be copied to a folder as well.

[[<DisplayType>] form is (<FormName> | inline)]  // Form to use on an 'Open'. Primary form is used by default.
    [when <SimpleCondition>]
    [<FormDefinition>]                             // valid on inline Form only
[drill list is <DrillListName>]
[create action is [<StateName>.]<ActionName>…]
[open 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 | enable) action [<StateName>.]<ActionName>…]     // not valid on AuditList
    [when <Condition>]

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

[suppress standard toolbar]                // by default standard toolbar will show unless CalledOutActions are defined
[disable all actions]
[called out actions only]
[disable field options]

[Actions]
    <CalledOutAction>…

[Quick Entry Fields]
    (<FormField> | <FormButton> | <CheckControl> | <FormText> | <ManualRepresentation>)…

[Detail Fields [(Below | On Side)]]        // default is 'Below'
    (<FormField> | <FormButton> | <CheckControl> | <FormText> | <ManualRepresentation>)…
```

```
        [(valid | visible) when <SimpleCondition>]

[Detail Sections]
    [show single panel tab]
    [print width is <Percent>]
        [(indent | right align | center | left align)] // default is indent
    ( <DetailSection>
    | <DetailSectionName> is a (DashBoardView | MultiListView))
        [mouse over text is <Message>]
        [title is <Message>]
        [(valid | visible) when <SimpleCondition>]
        [show (3 | 4) columns]                          // default is 3 columns
        [use 12 column grid]
        [print as dashboard]                            // valid only on DashBoardView
            [compress columns]
        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>]]
        [(show | hide) legend]                          // default is hide for PieChart; otherwise default is show
        [(show | hide) labels]                          // valid only for PieChart, default is show
        [show as donut]                                 // valid only for PieChart
            [center field is <RelatedField>]
                [(label is (<RelatedField> | <Message>) | no label)] // default is Field Label
                    [<TextStyle>…]
                [<FieldStyle>…]
        [show bars (vertically | horizontally)]         // valid only for BarChart, default is vertically
        [show elements(side-by-side | stacked)]         // valid only for BarChart, default is side-by-side
        [show area]                                     // valid only for LineChart
        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>]
                [link is <UILink>]

    [<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 an AggregationTable]
    [title is <Message>]
    [is default]
    [print form is <FormName>]                    // allows for a form to be used as the print layout
        [total form is <FormName>]
        [when printing detail sections]
    [visible when <SimpleCondition>]
    [suppress blank rows]
    [show totals (first | last)]                  // default is 'first'
    [Subtotal By]
        <RelatedField>…
            [page break]
    [Dimensions]
        <RelatedField>…
            [is default sort order [descending]]
            [(ascending | descending)]
            [label is <Message>]
            [link is <UILink>]
            [column width is <Number>]
            [caption is (representative text | <Message>)]
            [dimension valid when <SimpleCondition>]
                        // Allows the dimension to be ignored altogether –both for column display and pivot order.
                        // SimpleCondition must be based on the incoming context fields of this list rather than a
                        // particular instance of this business class. If the value of  SimpleCondition is indeterminate (it is
                        // not based strictly on valid context fields of this list invocation) the dimension will be considered invalid.
    Measures
        ((sum | avg | min | max) <RelatedField> | instance count)…
            [is default sort order [descending]]
            [label is <Message>]
            [column width is <Number>]
            [link is <UILink>]
            [link is <AggregationDrillLink>.<ListName>]
            [add filter [using (and | or) ]condition]          // default is 'and'
                where <Condition>
            [display <Number> decimals]
            [currency symbol is <RelatedValue>]                // default is 'leading'
                [first line and totals only]
            [show percent symbol]
                [first line and totals only]
            [display blank when zero]
            [display negative amount using (minus | parens)]
            [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 a particular instance of this business class. If the value of
                        // SimpleCondition is indeterminate (it is not based strictly on valid context fields of this list invocation)
                        // the column will not display.

[<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 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>]
                    [link is <UILink>]

DetailSection ::=
    [<DetailSectionName>]
        [mouse over text is <Message>]
        [title is <Message>]
        [implements ManualLoad]
            [message is <Message>]
                [description is <Message>]
                [icon is <Icon>]
                load button text is <Message>
        [(valid | visible) when <SimpleCondition>]
        [(valid | visible) for dimension <DimensionField>]
        [(double width | triple width | quad width)]     // valid only on DashBoardView
        [span (3 | 4 | 6 | 12) columns]                  // valid only on DashBoardView with 12 column grid
        [double height]                                  // valid only on DashBoardView
        [show in column(1|2|3|4|5|6|7|8|9|10|11|12)]     // valid only on DashBoardView
        [show <#> lines]                                 // valid only on List in DashBoardView – default is 5 lines
        (<Form> | <List> | link is <HttpURL> | card view is <CardViewName>)
            [<FormDefinition>]                           // valid only on inline Form
        [allow nested detail sections]                   // valid only on a list section
        [overview navigation is <NavigationName>]
            [label is <Message>]
        [page break]                                     // used only when printing to PDF

ListDisplayField ::=
    (<FormField> | <FormButton> | <CheckControl> | <FormText> | <ManualRepresentation>)…
        [is default sort order [descending]]       // valid in ColumnarView only
        [sort order is (<SetName> | primary)]
            [is default [descending]]
            [group by <FullFieldName>]             // FullFieldName must be a field in SetName
            [Subtotal By]                          // default is all fields in sort order except last one
                <RelatedField>…                    // must be field in sort order
                    [page break]
        [column width is <Number>]
        [total]
        [visible when <SimpleCondition>]
        [column visible when <SimpleCondition>]
```

```
        [suppress value retrieval when not visible]
                    // 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 (it is not based strictly on valid context fields of this list invocation) the column will not display.
    [link is <AggregationDrillLink>.<ListName>]
    [suppress when in dimension <DimensionField>]          // valid in a CubeView only
    [disable copy]                                         // valid for InlineCreate only
```

Card View

```
CardViewDefinition ::=
    <CardViewName> is a CardView
        [(show | hide) labels]          // hide labels is the default
        [raise <AlertType> alert when <SimpleCondition>…]
            [mouse over text is <Message>]
            [link is <UILink>]
        [title is <Message>]
            [no wrap]                                  // default is to wrap title
        [left column is  ( <RelatedValue>              // RelatedValue must be an image
                         | <Icon>
                         | representative image
                         | random background )]
            [foreground text is <Message>]
            [display as (portrait | photo | full) image]        // default is 'display as portrait'
            [missing image is (<ImageName> | random background)]
                [foreground text is <Message>]
        main column
            (<FormField> | <FormText> | <ManualRepresentation>)…
                [visible when <SimpleCondition>]
            [button row]                               // deprecated
                <FormButton>…
        [right column]
            [(left | right) align]                     // default is right align
            (<FormField> | <FormText> | <ManualRepresentation>)…
                [visible when <SimpleCondition>]
            [button row]                               // deprecated
                <FormButton>…
        [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]
            [when <SimpleCondition>]
        [is drill target]
            [when <SimpleCondition>]
        [propagate as a drill]
        [use for (<DisplayType> | action [<StateName>.]<ActionName>)]…
        [title is <Message>]                          // defaults to FormName

        [suppress standard toolbar]                   // by default standard toolbar will show unless CalledOutActions are defined
        [disable all actions]
        [called out actions only]
        [disable field options]

        [print form is <FormName>]                    // allows for a different form to be used when printing to pdf

        [Actions]
            <CalledOutAction>…
        [(restrict | enable) action [<StateName>.]<ActionName>…]    // valid on configuration only
            [when <Condition>]
        ( Layout
            <FormLayout>…
        | Manual Representation
            representation name is <Literal>
            Property Mapping
                <MessageProperty> = <RelatedValue>…
        )
```

Base Definitions

```
FormButton ::=
    button of ( <Message> [with (<Icon> | <ActionIcon>)]
                    [<TextStyle>…]
              | <Icon>
              | <ActionIcon>
              )
        is primary
        [(valid | visible) when <Condition>]
        [(link is <UILink>
         | sub form is (<FormName> | primary | inline)
            [<FormDefinition>]        // valid on inline Form only
         | <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 on inline Form only
         | <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> | <StandardAction>)
         | show editor for <RelatedField>             // Can be a BinaryDocument, Text, GroupField, or ArrayField
            using mime type <RelatedValue>
         | show preview of <RelatedField>             // Must be an Image
         | view <RelatedField>                        // Must be a BinaryDocument or Text
            using mime type <RelatedValue>
```

```
        | export <RelatedField>                          // Must be a BinaryDocument or Text
      )]…
          [when <SimpleCondition>]                       // Must have a condition if defining multiple targets
          [(restrict | enable) action [<StateName>.]<ActionName>…]   // valid on form invoke only
      [<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>]
          [link is <UILink>]
      [hidden]
      [when button pressed]
          ( validate (this field | <RelatedField>)       // RelatedField must be a mutable field on a form or list
          | <RelatedField> = <RelatedValue>              // RelatedField must be a mutable field on a form or list
          | refresh <RelatedField>                       // RelatedField must be on a form or list
          | refresh dependent panels
          )…
              [when <SimpleCondition>]

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>…]
        [mouse over text is <Message>]
        [suppress label]
        [<FieldStyle>…]
        [align as label]                                 // make field line up in label column – valid only with no label
        [form is <FormName>]                             // valid on Group and Array fields only
        [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
        [compact format [(only | button of <Message>)]]
        [display as select]                              // overrides implements UserDefinedStates
        [select is [<RelatedLink>.]<ListName>]           // valid on key field only
        [default filter operator is (contains | starts with | equals)]
                                                         // valid only on alpha field types
        [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)]
        [compare with <RelatedField>]                    // this field and RelatedField must be of type LPL
        [time zone is <RelatedField>]                    // valid only on a TimeStamp field; <RelatedField> must be a TimeZone
        [display <Number> decimals]
        [currency symbol is <RelatedValue>]
            [first line and totals only]
        [show percent symbol]
            [first line and totals only]
        [display negative amount using (minus | parens)]
        [display as (hijri | gregorian) date]
        [display as [(portrait | photo | full)] image [up to <Literal> lines]] // default 'display as portrait'
            [missing image is (<ImageName> | random background)]
                [foreground text is <Message>]           // default number of lines is 8
```

```
        [image is <FullFieldName>]      // RelatedField could be an Array of images or an Array of GroupFields that contain an image
        [title is <FullFieldName>]      // if Array of GroupFields must specify image field on GroupField and may specify title
    [display as switch]                 // valid a Boolean type field only
    [display as tag]
        [color of <Color> [when <SimpleCondition>]…]
    [display as rating]                             // must be a Numeric or Decimal 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>]
            [link is <UILink>]
    [display as ((meter | thermometer) chart | progress bar) [up to <Literal> lines]]
                                                    // default number of lines is 8 unless this is a progress bar which can have 1 line
         [meter range is <RelatedValue>]            // not required when Percent type field
        [<AlertType> alert range is from <RelatedValue> to <RelatedValue>…]
            [mouse over text is <Message>]
            [link is <UILink>]
    [raise <AlertType> alert when <SimpleCondition>…]
        [mouse over text is <Message>]
    [show as <StatusIcon> when <SimpleCondition>…]
        [mouse over text is <Message>]
        [link is <UILink>]
    [display when blank]                            // On a SummaryForm the default is to not display blank fields
    [display zero]                                  // display '0' if value is blank
    [display blank when zero]                       // display blank if value is zero (some field types default to show zeros)
    [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>]
    [display as [horizontal] radio buttons]                 // must be a States or UserDefinedStates field
    [display (date | time | hours and minutes)]              // must be a TimeStamp or Time field
    [display with time zone]                                // must be a TimeStamp
    [<States>]
    [display as required [when <SimpleCondition>]]
    [hidden]                                                // valid on configuration only
    [required]                                              // valid on configuration only
    [initial value is <RelatedValue>]                       // valid on configuration only
    [constraint <Condition>]                                // valid on configuration only
        [<Message>]
    [depends on <RelatedField>[, <RelatedField>]…]          // RelatedField must be a mutable field on a form or list
        derived value is <RelatedValue>
    [when value changed]
        ( validate (this field | <RelatedField>)            // RelatedField must be a mutable field on a form or list
        | <RelatedField> = <RelatedValue>                   // RelatedField must be a mutable field on a form or list
        | refresh <RelatedField>                            // RelatedField must be on a form or list
        | refresh dependent panels
        )…
            [when <SimpleCondition>]

FormLayout ::=
    <LayoutDirective>
        [( <FormField>
         | <FormButton>
         | <CheckControl>
         | <CaptchaControl>
         | <ManualRepresentation>
         | <FormText>
         | <MultiSelectField>
```

```
          | <FormLayout>…
        )]…

SectionLayout ::=
    <SectionDirective>
        <FormLayout>…

SectionDirective ::=
    section of <SectionName> [visible when <SimpleCondition>]
        [label is <Message>]

ColumnOptions ::= [grid] [distributed] [independent] [table] [disable horizontal alignment]

LayoutDirective ::=  ( paragraph [centered] [distributed]
                     | single column [independent]
                     | two column   <ColumnOptions>
                     | three column <ColumnOptions>
                     | four column  <ColumnOptions>
                     | five column  <ColumnOptions>
                     | six column   <ColumnOptions>
                     | seven column <ColumnOptions>
                     | eight column <ColumnOptions>
                     | nine column  <ColumnOptions>
                     | ten column   <ColumnOptions>
                     | column headers                  // valid only with '<n> column table'
                     | row headers                     // valid only with '<n> column table'
                     | column(1|2|3|4|5|6|7|8|9|10) [of <Percent>]
                     | <Header>
                     | (valid | visible) when [<BusinessClass>.]<SimpleCondition>
                                                       // must specify <BusinessClass> when in MenuLayout
                     | locale of <RelatedValue>
                     | display as text
                     | blank line
                     | page break
                     | footer
                     | large menu
                     | small menu                      // default
                     | <Form>                          // must be top level layout directive
                     | <List>                          // must be top level layout directive
                    )

MultiSelectField ::=
    multi-select field is <FullFieldName>                          // valid on key field or array of key fields only
        [(label is (<RelatedField> | <Message>) | no label)]       // default is Field Label
            [<TextStyle>…]
        [mouse over text is <Message>]
        [display field is <RelatedField>]
        [select is [<RelatedLink>.]<ListName>]                     // valid on key field only
        [use <Literal> lines]                                      // default is 5 lines
        [display as required [when <SimpleCondition>]]
        [hidden]                                                   // valid on configuration only
        [required]                                                 // valid on configuration only
        [initial value is <RelatedValue>]                          // valid on configuration only
        [constraint <Condition>]                                   // valid on configuration only
            [<Message>]
        [when value changed]
            ( validate (this field | <RelatedField>)               // RelatedField must be a mutable field on a form or list
            | <RelatedField> = <RelatedValue>                      // RelatedField must be a mutable field on a form or list
```

```
        | refresh <RelatedField>                          // RelatedField must be on a form or list
        | refresh dependent panels
        )…
            [when <SimpleCondition>]
```

Action

```
ActionFormDefinition ::=
    <FormName> is an ActionForm
        [use for <DisplayType>]
        action is [<StateName>.]<ActionName>…
            [on completion navigate to <NavigationName>]
                [when <SimpleCondition>]
                    // 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> | <SectionLayout>)…
```

Composite

```
CompositeFormDefinition ::=
    <FormName> is a (CompositeForm | WizardForm)
        [is primary]
            [when <SimpleCondition>]
        [is drill target]
            [when <SimpleCondition>]
        [summary form is <FormName>]                    // valid on WizardForm only – acts as a 'commit' boundary
        [propagate as a drill]
        [use for (<DisplayType> | action [<StateName>.]<ActionName>)]…
        [title is <Message>]                            // defaults to FormName
        [print form is <FormName>]                      // allows for a different form to be used when printing to pdf
        [context form is <FormName>]                    // allows common 'context' fields to be put outside the panels–not valid on Wizard
            [show context form on left]
            [background is (<RelatedValue> | theme color)]      // RelatedValue must be an image
        [implements IndependentFormPanels]
        [implements DashBoardView]                      // valid on CompositeForm only
            [show context form on top]
            [show (3 | 4) columns]                      // default is 3 columns
            [use 12 column grid]
            [print as dashboard]
                [compress columns]
        [implements CartView]                           // valid on CompositeForm only
            cart is <RelatedLink>                       // must resolve to an instance of this BusinessClass
        [show panel control on left]
        [show panel navigation]
            [linear when <SimpleCondition>]             // similar to WizardForm
            [last panel navigation is <UILink>]
                [next button title is <Message>]
```

```
        [show steps]                                    // valid on WizardForm only – displays Steps Graphic
            [label is <Message>]

        [suppress standard toolbar]                     // by default standard toolbar will show unless CalledOutActions are defined
        [disable all actions]
        [called out actions only]
        [disable field options]

        [orientation is (portrait | landscape)]
        [font offset is <Number>]

        [Actions]
            <CalledOutAction>…
        [(restrict | enable) action [<StateName>.]<ActionName>…]    // valid on configuration only
            [when (<Condition>)]
        (<CompositeFormPanel> | <CompositeFormMultiListPanel>)…

CompositeFormPanel ::=
    <PanelName> is a DashBoardPanel…
        [show (3 | 4) columns]                          // default is 3 columns
        [use 12 column grid]
        [title is <Message>]
        [icon is <Icon>]
        [(valid | visible) when <SimpleCondition>]
        [mouse over text is <Message>]
        [print as dashboard]
            [compress columns]
        <CompositeFormPanel>…

    <PanelName> is a Panel…
        [title is <Message>]
        [icon is <Icon>]
        [(valid | visible) when <SimpleCondition>]
        [mouse over text is <Message>]
        [panel header is <Message>]
        [overview navigation is <NavigationName>]               // valid in a DashBoardView context only
            [label is <Message>]
        [(double width | triple width | quad width)]
        [span (3 | 4 | 6 | 12) columns]
        [double height]
        [show in column(1|2|3|4|5|6|7|8|9|10|11|12)]
        (<FormPanelDefinition> | <FormSubPanelDefinition>)

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

```
FormSubPanelDefinition ::=
    [visited action is [<StateName>.]<ActionName>          // valid when 'show panel navigation' is on
    [in progress when <SimpleCondition>]                   // valid when 'show panel navigation' is on
    [completed when <SimpleCondition>]                     // valid when 'show panel navigation' is on
    [raise <AlertType> alert when <SimpleCondition>…]      // valid when 'show panel navigation' is on
        [mouse over text is <Message>]
        [link is <UILink>]
    sub panels are <RelatedLink>
        title is <Message>
        [(valid | visible) when <Condition>]
        [mouse over text is <Message>]
        [display when <Condition>]…                        // last panel in set cannot have 'display when'; all others must
            <FormPanelDefinition>
        <FormPanelDefinition>

CompositeFormMultiListPanel ::=
    <PanelName> is a MultiListPanel…
        [title is <Message>]
        [(valid | visible) when <Condition>]
        [mouse over text is <Message>]
        (<MultiList> | <MultiListLayout>)

MultiList ::=
    <List>…
        [visible when <SimpleCondition>]
        [panel text is <Message>]
        [show <#> lines]                                   // defaults to 5 lines
        [<FormPanelActions>]

MultiListLayout ::=
    Layout
        <MultiListLayoutDirective>…
            (<MultiList> | <MultiListLayoutDirective>)…

MultiListLayoutDirective ::=
    ( single column
    | two column
    | column(1|2)
    | (valid | visible) when [<BusinessClass>.]<SimpleCondition>
                                                           // must specify <BusinessClass> when in MenuLayout
    | locale of <RelatedValue>
    )

FormPanelActions ::=
    [Special Actions]                                      // allows actions from the primary business class to be called from this context
        <CalledOutAction>…
```

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                    // defaults to primary list
    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>]                                 // defaults to MatrixName
        [display negative amounts using parens]
        [display amounts in (thousands | millions)]
            [show in title]

        [row search field is <FullFieldName>]                // FullFieldName from the context of the row's business class

        [implements PeriodFilter]
            ( date is <FullFieldName>                        // FullFieldName from the context of the column's business class
            | (monthly | weekly | daily) period is <FullFieldName>
                year is <FullFieldName>
            )
            [show (day | week | month | quarter | year)]     // defaults depending on filter type
            [current (date | period) is <RelatedValue>]      // RelatedValue from the context of this business class
                current year is <RelatedValue>

        context form is (<FormName> | inline)
            [<FormDefinition>]                               // valid on inline Form only

        Column
            Display Fields
                ( <FormField>…
                | <FormButton>…
                | <CheckControl>…
                | <ManualRepresentation>…
                )

        Cell
            [overview navigation is <NavigationName>]
```

```
        [label is <Message>]
    Display Fields
        ( <FormField>…
            [(valid | visible) when <SimpleCondition>]
        | <FormButton>…
        | <CheckControl>…
        | <ManualRepresentation>…
        )

    Detail Fields
        ( <FormField>…
        | <FormButton>…
        | <CheckControl>…
        | <ManualRepresentation>…
        )
```

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>"
                    header2
                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
        [suppress header]
        [suppress footer]
        [disallow show as pdf]
        [orientation is (portrait | landscape)]
        [font offset is <Number>]

        [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>
    | <ManualRepresentation>
    | <FormText>
    | form is [<RelatedLink>.]<FormName>          // can take only an OTM RelatedLink
        [<Header>]                                // if RelatedLink is empty then header will not display
    | list is [<RelatedLink>.]<ListName>          // can take only an OTM RelatedLink
        [show grid lines]
        [<Header>]                                // if RelatedLink is empty then header will not display
    | <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 SplashPage
    [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…                 // multiple panels would flow on the page inline rather than have tabs
        [title is <Message>]
        [mouse over text is <Message>]
        <MenuPanel>
        [show in column(1|2|3|4|5|6|7|8|9|10|11|12)]

<PageName> is a Page
    [title is <Message>]                                // defaults to PageName
    [allow anonymous access]
    [show panel control on left]
    [orientation is (portrait | landscape)]
    [font offset is <Number>]

    [cart form is <BusinessClass>.<FormName>]           // FormName must be a CompositeForm that implements CartView

    [Context Form]
        [background is (<ImageName> | theme color)]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>]
        [show (3 | 4) columns]                          // default is 3 columns
        [use 12 column grid]
        [print as dashboard]
            [compress columns]
        <PaneName> is a Pane…
            [title is <Message>]                        // defaults to PaneName
            [(double width | triple width | quad width)]
```

```
            [span (3 | 4 | 6 | 12) columns]
            [double height]
            [show in column(1|2|3|4|5|6|7|8|9|10|11|12)]
            <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 MultiListPanel …
        [title is <Message>]
        [mouse over text is <Message>]
        [(valid | visible) when <SimpleCondition>]
        <ListPanel>…

    <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>…

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.

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

Panels

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

List

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

```
                )
        [visible when <SimpleCondition>]            // valid on a MultiListPanel only
        [panel text is <Message>]                   // valid on a MultiListPanel only
        [show <#> lines]                            // valid on a MultiListPanel only
        [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 on a CubeView only, overrides the CubeView Row Dimension
                <DimensionField>…
            [overview navigation is <NavigationName>]          // valid on a DashBoardPanel only
                [label is <Message>]
            [<ListDefinition>]                      // valid on inline List only
        [helper list is <RelatedLink>.(<ListName> | primary)…]
                                                    // valid on a relative pane only - <RelatedLink> is relative to pane<PaneNumber>
            [valid when <SimpleCondition>]
            [title is <Message>]
            [<Action>]
                [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>]
            )…
```

Menu

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

MenuLayout ::=
    ( <FormText>
    | <MenuItem>
        [title is <Message>]
        [menu item name is <MenuItemName>]
        [description is <Message>]
        [(icon is <Icon> | image is <ImageName>)]
        [allow anonymous access]
    | <LayoutDirective>
        [<MenuLayout>…]
    )
```

Search

```
SearchPanel ::=
    search form is <BusinessClass>.<FormName>
        [list is (<ListName> | inline)]             // defaults to primary list on resultant business class
            [<ListDefinition>]                      // valid on inline List only
        [Context]
```

```
            <FieldName>…
        [Actions]
            ( [action is] [<StateName>.]<ActionName>…
                [<Label>]
            | form is <RelatedLink>.<FormName>…      // <RelatedLink> can only be one of the context fields
                [<Label>]
            )
// 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
```

URL

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

Work View

```
WorkViewPanel ::=
    ( business class is  ( <BusinessClass>[.<RelatedLink>]
                         | pane<PaneNumber>.<RelatedLink>
                         | <Agent>.<RelatedLink>
                         )
        [title is <Message>]                        // defaults to the title of <ListName>
        [show <#> lines]                            // 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]                        // This is additive to the list definition's (<ListName>) Instance Selection
            where <Condition>
    | detail fields view of pane<PaneNumber>
    | link display view of pane<PaneNumber>
    )
```

Example of Page definition

```
BuyerPage is a Page
    Worksheet is a MultiPanePanel
        pane1
            business class is SourcingEventWorksheet
                Context
                    Actor required

                Actions
                    CreateEvent
                    SendToPO
                    Delete
        pane2
            business class is PoInterface
                Context
                    Pocompany

                Actions
                    CreateEventWorksheet

    Draft is a Panel
           business class is SourcingEvent
            list is DraftEvents  // only show create events within Draft state
            Context
                Company required

    Open is a Panel
        business class is SourcingEvent
            list is OpenEvents  // should not show create events
            Context
                Company required

    Awarded is a Panel
        business class is SourcingEvent
            list is AwardedEvents
            Context
                Company required
```

Menu Definition

// Menu definitions are in the same name space as BusinessClasses, BusinessTasks, and Pages. A Menu definition file has an extension of .menu

Menu Structure ::=

```
<MenuName> is a Menu
    [show as springboard]              // bring up the menu in a springboard
    [Search Terms]
        <Message>…

    (<MenuItems> | <MenuPanel>)        // If a menu is used in a Navigation Bar within a .wepapp file, then the first menu item is 'Home'
                                       // which takes the user to the 'home page' in the webapp definition file.
```

```
    [<MenuName> is a Menu …]            // This allows menus to be 'in-lined' within a single menu definition file
        [Search Terms]
            <Message>…
        <MenuItems>

MenuItems ::=
    Menu Items
        <MenuItemName>…
            [title is <Message>]
            [(icon is <Icon> | image is <ImageName>)]
            [(valid | visible) when <BusinessClass>.<SimpleCondition>]
            <MenuItem>
            [allow anonymous access]
            [Search Terms]
                <Message>…

MenuItem ::=
    ( page is <PageName>[.<PanelName>]
    | list is <BusinessClass>[<AsOfOperator>].(<ListName> | primary)
    | form is <BusinessClass>.(<FormName> | primary)
        [actor agent required]
        [create action is [<StateName>.]<ActionName>]
        [open action is [<StateName>.]<ActionName>…]
        [(restrict | enable) 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.

Logic for 'form is <BusinessClass> ...'

```
if (actor.agent(<BbusinessClass>) exists)
    open form using open action
else
if ("actor agent required")
    throw error message
else
    set <BusinessClass> keys from context variables (actor context vars and session.key vars)
    if (<BusinessClass> exists)
        open form using open action
    else
        open form using create action
```

Logic for 'action is <BusinessClass> ...'

```
if (action type is Set Action or Create Action)
    run action
else
if (actor.agent(<BusinessClass>) exists)
    run action against actor.agent(<BusinessClass>)
else
    set <BusinessClass> keys from context variables (actor context vars and session.key vars)
    if (<BusinessClass> exists)
        run action against <BusinessClass> instance
    else
        throw error
```

Webservice Interface Definition

// A Webservice Interface definition is defined within a module and has an extension of .wsi
// This definition is required to execute a Webservice operation from an lpl definition.

Webservice Interface Structure ::=

```
<WebserviceInterface> is a WebserviceInterface
    wsdlURL is <Literal>                // for example - http://webservices.amazon.com/AWSECommerceService/AWSECommerceService.wsdl
    endpoint is <Literal>               // for example - http://soap.amazon.com/onca/soap?Service=AWSECommerceService
    service is <Literal>                // for example - AWSECommerceService
    service style is <WSIServiceStyle>  // for example - SOAP_V2
    authentication is <WSIAuthentication>  // for example - basic

    <ActionName> is a[n] <WSIActionType> Action …
        [method is <Literal>]
        Input Record namespace is <Literal>  // for example - http://webservices.amazon.com/AWSECommerceService/2006-11-14
            <FieldName> [<DataDefinition>]…
                [attribute name is <Literal>]
                [element name is <Literal>]
                [required]

        Output Record
```

```
            <FieldName> [<DataDefinition>]…
                [attribute name is <Literal>]
                [element name is <Literal>]
                [repeating]

WSIServiceStyle ::=  ( RESTGET
                     | RESTPOST
                     | SOAP_V1
                     | SOAP_V2
                     | XMLHTTP
                     | FACEBOOK
                     )
                          // These client types will process the interface synchronously:
                          // - RESTGET uses an HTTP GET and returns an XML structure.
                          // - RESTPOST uses an HTTP POST to submit an XML request that is 'one' level deep and returns an XML structure.
                          // - SOAP_V1 is the first version of SOAP, meaning RPC encoding, this type of client will not be supported moving forward.
                          // - SOAP_V2 is the second version of SOAP, supporting Document/Literal type encoding, the most common type of
                          //                  encoding Namespacing is supported.
                          // - XMLHTTP uses an XML request and receives an XML response. This type of client is to be used for proprietary web
                          //                  services that do not use SOAP.

WSIAuthentication ::= ( basic
                      | token
                      )
                          // These authentication types predict if the client needs to pass authentication info in the request
                          // - basic uses an HTTP GET and returns an XML structure.
                          // - token uses an HTTP POST to submit an XML request that is 'one' level deep and returns an XML structure.

WSIActionType ::=   ( Create            // All actions are processed synchronously
                    | Update
                    | Delete
                    | Find
                    | Instance
                    | List
                    )
```

Example of Webservice Interface

```
USZip is a WebserviceInterface

    wsdlURL is "http://www.webservicex.net/uszip.asmx?wsdl"
    endpoint is "http://www.webservicex.net/uszip.asmx"
    service is USZip
    service style is SOAP_V2

    GetInfoByZIP is an Instance Action
        Input Record namespace is "http://www.webserviceX.NET"
            USZip is Alpha size 20
                required

        Output Record
            GetInfoByZIPResponse is GroupField
                GetInfoByZIPResult is GroupField
                    City is Alpha size 100
                        element name is CITY
                    State is Alpha size 10
                        element name is STATE
                    Zip is Alpha size 10
                        element name is ZIP
                    AreaCode is Alpha size 10
                        element name is AREA_CODE
                    TimeZone is Alpha size 10
                        element name is TIME_ZONE
```

Example of Webservice Interface Invocation

```
BusinessPartner is a BusinessClass
…
    invoke GetInfoByZIP USZip
        invoked.USZip             = PostalCode
        BillingAddress.PostalCode = result.GetInfoByZIPResponse.GetInfoByZIPResult.Zip
        BillingAddress.Municipality     = result.GetInfoByZIPResponse.GetInfoByZIPResult.City
        BillingAddress.StateProvince    = result.GetInfoByZIPResponse.GetInfoByZIPResult.State
```

Security Class Definition

Security Class Structure ::=

```
<SecurityClassName> is a SecurityClass
    [extends <SecurityClassName>]
    [description is <Message>]
    Access Rights
        <PolicyDefinition>…
```

Structure Definitions

```
AccessPolicy ::=
    <SecurableObjectSpecification>
        <AccessRule>…

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

OntologicalPolicy ::=
    <KeyField> KeyField
        <GrantRule>…
```

Base Definitions

```
ActionKeyword ::=  ( all actions
                   | all audit views
                   | all creates                    // 'all creates' assumes 'all inquires' by default
                   | all deletes                    // use 'except' to deny 'all inquiries' if this is not intended
                   | all inquiries                  // separate list of objects with a comma if on the same line
                   | all functions
                   | all updates
                   | all global UI configuration    // valid on DataArea SecurableObjectType only
                   | all security configuration     // valid on DataArea SecurableObjectType only
                   | all personalization            // valid on DataArea SecurableObjectType only
                   | limited list personalization   // valid on DataArea SecurableObjectType only
                   | limited form personalization   // valid on DataArea SecurableObjectType only
                   | all scheduled actions          // valid on DataArea SecurableObjectType only
                   | all actor groups               // valid on DataArea SecurableObjectType only
                   | data menu                      // valid on DataArea SecurableObjectType only
                   | future data indicator          // valid on DataArea SecurableObjectType only
                   | UpdateEffectiveDatedData
                   | ViewAuditLog
                   | ViewFullAuditLog
                   )

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

Access ::= ( is accessible [and attachable]
           | is (not | neither) accessible [nor attachable])

AccessRule ::=
    <Access> [in [past][,][current][,][future]]
        <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>
                      | 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
                    )

LMObject ::= <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
```