Study-guide for Appian Training

Please **copy and paste** the below template for each entry.

Make a page break (without smashing ‘return’/‘enter’) using **Ctrl + Enter.**

**Topic**

Definition:

Purpose:

Implementation Example (what):

Implementation Example (how):

Alternate Rules:

Advantage/drawbacks:

Associated Topics:

**Related Resources...**

<https://docs.appian.com/suite/help/20.4/Glossary.html>

[**Topic 1**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.7bl3xpeq3zcw)

[**BPM 6**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.hkj0oehypwvi)

[**Intro to Appian 7**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.pxq2j4iltr5d)

[**Appian Data Types 8**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.z7d85qk8ngik)

[**Rule Inputs 11**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.kjvgwhdzz01k)

[**CDTs 12**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.xsjw2q4h65y)

[**CDT Best Practices 13**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.lxxjidgait1s)

[**XSD 14**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.xapb0627qucv)

[**Appian Smart Services 17**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.lf4qd7b3zvn3)

[**Rule Event 18**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.ixm1tfsompmx)

[**Interface editor 21**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.ysxzc87ek12b)

[**Interface Layouts 22**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.e78uzzlchqnn)

[**Interface components 23**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.9muoo5ol1d3b)

[**SAIL (Self-assembling Interface Layer) 24**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.p8zgnjcaznep)

[**SAIL Recipes 25**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.ysjb0lpugmp1)

[**Multiple Node Instances(MNI) 26**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.htkfji1ln9xo)

[**Records: 29**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.hpknzobnlt2e)

[**Default Filters: 30**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.1aoq03xzxmde)

[**Related Actions 32**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.tq8q49hbj842)

[**Entity-Backed Records 34**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.ej92ui9k6koy)

[**User Filters: 36**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.b7kh738tb31)

[**Database Smart Services 37**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.p72nif30w9gl)

[**Query Entities 38**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.oetikyv98phe)

[**Expression Rules 40**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.pmp5zqemaa8q)

[**Appian Designer Mode:Interface 41**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.g8ij5141myfe)

[**Dynamic Interfaces 42**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.98v77zypvz9z)

[**Security 44**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.65my2hydo2q8)

[**Appian Functions 45**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.3ojsldoqwri8)

[**Editable Grid 46**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.q5l2l842l0nf)

[**Read-Only Grid 47**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.avv0mxud5nhu)

[**Cascading Dropdown 48**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.vin4nh600hla)

[**Sites: 49**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.khnzl9lsoqu)

[**Start Form: 49**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.1gby3up159el)

[**Exception: 50**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.slcmtsylb56k)

[**Agile/Scrum 51**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.vbzlma2ye199)

[**Data Store Objects 52**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.bn35agtnkkif)

[**Data Store Entities 53**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.9tkb0v86h5rt)

[**Task report 54**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.wb2879dxkbpm)

[**Tempo 55**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.32tf576ojxam)

[**Designing Drillable Reports 57**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.rbiocalc3viy)

[**Linking Records and Reports 58**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.mu05dgnqdu08)

[**Application Patches 61**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.rtytp4gzeg0w)

[**Admin Console 62**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.2a6rx5d13khh)

[**Users 64**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.w6ajx9xjb030)

[**Local Variables using load() and with() 67**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.feh9ccyqh18e)

[**Process Management & Debugging 68**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.gfyqcp2vxas4)

[**Appian Web Service Integration 69**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.nou1rvnnspvh)

[**Web API 71**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.happf5vorfba)

[**Connected Systems 73**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.uk1j4rrceiz6)

**Type your name into the right-hand column for *topics* that you will cover.**

Please note:

* Blue/yellow is for every other day on the schedule. The color is merely for your reference.
  + Not all “days” are created equal.
* Feel free to have multiple people working on the same topic; we’ll cover the topics more thoroughly if we have more than one perspective on each of these topics.

|  |  |
| --- | --- |
| Topic(s): | Assigned to: |
| BPM | Robin Friedman, Karl Jean |
| Agile/Scrum | James Smith |
| Intro to Appian | Robin Friedman |
| Appian data types | Anthony Nguyen |
| Rule inputs | Anthony Nguyen |
| CDTs | Anthony Nguyen |
| XSD files | Robin Friedman |
| CDT best practices | Robin Friedman |
| Data stores | James Smith |
| Data entities | James Smith |
| Interface editor | Stroh Leslie |
| Interface components | Stroh Leslie |
| SAIL (Self-assembling Interface Layer) | Stroh Leslie |
| SAIL “Recipes” | Stroh Leslie |
| Appian Designer Mode: Interface | Trevor Roy |
| Dynamic interfaces | Trevor Roy |
| Expression Editor | Tope |
| Process Modeling | Tope |
| Process Modeling best practices | Tope |
| Groups (security?) | Tope |
| Write to (multiple) data sources | Christian Harris |
| Query entities | Christian Harris |
| Database Smart Services | Christian Harris |
| Smart Services (broadly) | Christian Harris |
| Multiple node instances (MNI) | Karl Jean |
| Security | Trevor Roy |
| Appian functions | Trevor Roy |
| Paging (read-only) grid | Pranav |
| Editable grid | Pranav |
| Cascading dropdown | Pranav |
| Send email (Smart Service) | Pranav |
| Intro to Records | Jason Yip |
| User filters | Jason Yip |
| Default filters | Jason Yip |
| Related actions | Anthony Rodriguez |
| Entity-backed records | Anthony Rodriguez |
| Sites | Bryan Mejia |
| Start form | Bryan Mejia |
| Exception | Bryan Mejia |
| Escalation | Jon Baker |
| Rule Event | Jon Baker |
| Start process | Solomon |
| Task report | Solomon |
| Tempo | Anthony Rodriguez |
| Designing drillable reports | Devin Bell |
| Linking records and reports | Devin Bell |
| Sub-process | Solomon |
| Import/Export | Solomon |
| Application patches | Jahdiel Evans |
| **Week 8 & onwards** |  |
| Administration console | Jahdiel Evans |
| Users | Jahdiel Evans |
| Local variables using load() and with() | Tristan Garcia |
| Process management & debugging | Kyle Stolle |
| Application management |  |
| Appian engines |  |
| Appian web service integration | Kyle Stolle |
| Web API | Tristan Garcia |
| Connected Systems | Tristan Garcia |
| Shared Components | ; |
| Plugins |  |
| Plugin Development |  |
| Function plugins |  |
| Servlet plugins |  |
| Deploying/using custom plugins |  |
| Smart Services plugins |  |
| Plugin best practices |  |
| Deployment to production |  |
| Appian Health Check |  |
| Export to Excel |  |
| Generate PDF |  |

**BPM**

Definition:

* Stands for Business Process Management.
  + (BPMN is a distinct initialism meaning Business Process Model Notation)
  + Design → Model  →  Execute → Manage →  Optimize cycle
* From the Glossary: “Business Process Management (BPM) is a **change management and system implementation methodology** to aid the continuous comprehension and management of business processes that interact with people and systems, both within and across organizations. It is a methodology based on the following **assumptions** —
  + business processes are ever-changing and developing,
  + processes cross-cut each other, and
  + processes must flow between multiple organizations and interested parties.”

Purpose:

* “Design, execute, manage and optimize your processes — all with the Appian low-code automation platform.”
* Use [Tempo](https://www.youtube.com/watch?v=AbSOvrYTmI4) for collaborative BPM.

Implementation Example (what):

Implementation Example (how):

Alternate Rules:

Advantages/Drawbacks:

* Advantages
  + Improved Business Agility
  + Reduced Costs and Higher Revenues
  + Higher Efficiency
  + Better Visibility
  + Compliance, Safety, and Security

Associated Topics:

Business Process Automation(BPA)

**Intro to Appian**

Definition:

* Low-code Automation Platform
* Full-stack automation for enterprise-level applications

Purpose:

* Unified,
  + Integrates **22 systems into 1** Appian-based interface.
* Flexible,
  + Appian **works with many services** (e.g. Salesforce, Oracle, DocuSign),
  + You can also make your own custom APIs
* Fast,
  + **Low-code** means 20x **faster development and deployment**.
  + Drag-and-drop, point-and-click allow for easy and fast iteration.
  + Business processes can be drawn (visually constructed using BPMN)
  + **Reusable components** allow creating each successive app faster
  + **Prebuilt UI configurations** (also, look good)
  + Automatically **mobile-friendly** instead of making native iOS and Android apps separately
  + **Future-proof** with automatic upgrades to support future operating systems, browsers, devices, and UI tech
* Worry-free
  + Greatest number of **security certifications**
  + **Scalable** solution
  + **Reliable** devops tools
  + **Compliant** in customer data protection, regulatory requirements
  + **Oversight** and audit trails for CIOs from largest companies...

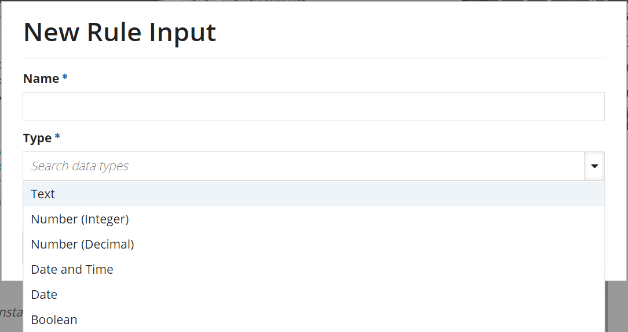
Implementation Example:

* [Merck chose Appian](https://www.youtube.com/watch?v=MLYMPWr57BE) for its information management system. Databases, solutions... integrates information from what are otherwise separate silos-- people can see the big picture. Flexible on devices. Looking for knowledge is easier now with Appian, freeing up workers to focus on bringing medications/wellness products to patients.
  + Appian is used across different industries by large enterprises.
* Appian helps developers through the Documentation, Academy, Playbook, UX Design Guide, and Community

Associated Topics:

* [BPM](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.oqg3mtyknbr9)

**Appian Data Types**

* Definition
  + The specific types of formats by which data in Appian conforms to
  + Appian data types can be one of the system types or a custom data type built from an XSD, Java object, or imported from a WSDL by the Call Web Service smart service
* Purpose
  + Classifying data in order to give a variable or object a set of compatible operations to work with without error
* Implementation (Example)
* 
* Alternative Rules
* Steps to implement
  + When creating a rule input, you will specify the desired data type it should hold
  + Each data type will have specific functions that they are compatible with
  + It is important to know what need you are trying to satisfy before defining a data type
* Main categories of data types:
  + Any Type
    - Generic data type that will accept data of any data type
    - Only available for use as an expression input for rules and certain expression functions, such as the if() function
  + Map Type
    - Used for associative arrays that store data in key-value pairs
  + Primitive System Data Types
    - Boolean
      * Default value is null
      * Values include true and false
      * toBoolean() type casting will evaluate 0 as false and all other numbers as true
      * toBoolean() type casting can also convert true and false in text form to a Boolean value
    - Date
      * Default value is an empty value
      * Values are dates by default formatted in MM/DD/YYYY
      * Minimum value of 1/1/1000 and a maximum value of 12/31/9999
      * Does not accept text input
      * Numerical and text values can be converted with the todate() function
    - Date and Time
      * Saved in GMT and then converted to the end user’s time taking daylight savings into account
      * Numerical and text values can be converted with the todatetime() function
    - Encrypted Text
      * Cannot be cast into any other type and no type can cast into it
      * Value is only decrypted when displayed as a value in an EncryptedTextField or within a plug-in using the EncryptionService public API
    - Number (Decimal)
      * Default value is 0.0
      * Can be cast from text strings with the todecimal() function
    - Number (Integer)
      * Default value is 0, and the null value is -2^31
      * Can be cast from text strings with the tointeger() function
    - Text
      * Used to store any UTF-8 string
      * Numerical values can be entered, but they will be interpreted as a string until cast
    - Time
      * Data created using the time(hour, minute, second) function
      * Cannot accept a text string input
      * Not adjusted to time zones when saved or displayed
  + Complex Data Types
    - Data types that cannot be edited or deleted
    - The following are complex data types made available to support smart services:
      * EntityData
      * EntityDataIdentifiers
      * HealthCheckOutput
      * LabelValue
      * LabelValueTable
      * ListViewItem
      * Facet
      * FacetOption
      * ObjectTestResult
      * PagingInfo
      * ProcessInfo
      * Query
      * SortInfo
      * TestCaseResult
      * TestRunResult
      * Writer
* Associated Knowledge
  + [CDT (Custom Data Types)](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.xsjw2q4h65y)
  + [Rule Inputs](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.kjvgwhdzz01k)

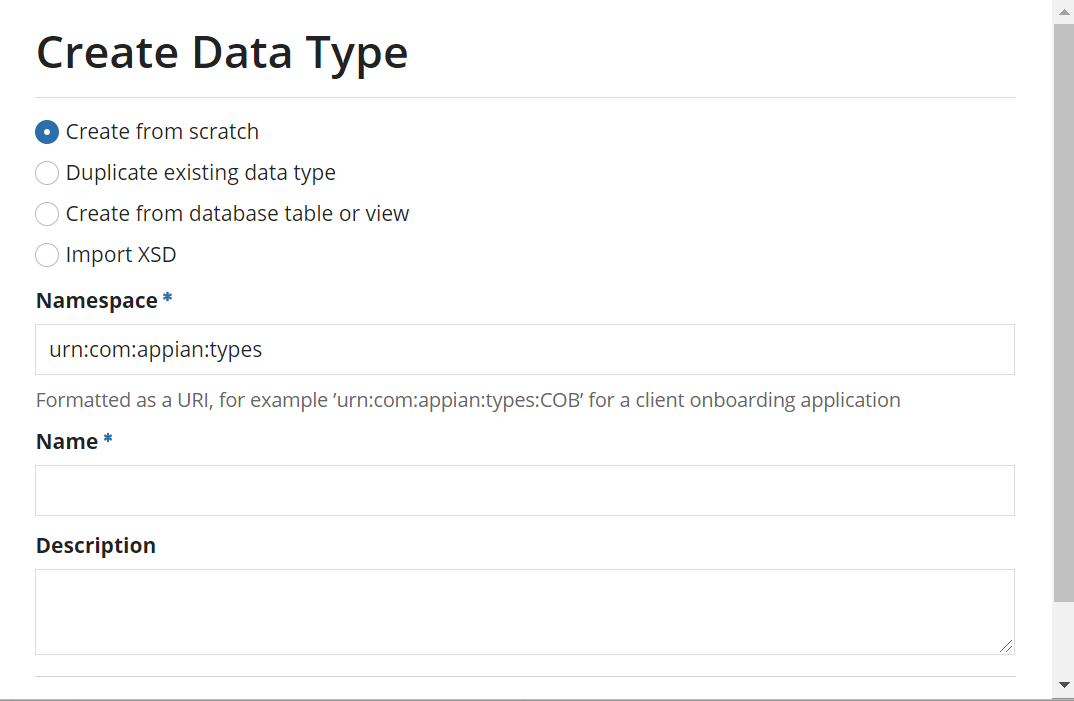
**Rule Inputs**

* Definition
  + A intermediary structure to take in a user input in order to pass data on to an expression
* Purpose
  + Rule inputs are used to pass data into an expression rule
* Implementation (Example)
* 
* Alternative Rules
* Steps to implement
  + In an interface, open the rule input tab to view existing rule inputs for the interface
  + Click on the “+” in the tab to create a new rule input
  + You can give the rule input a customized name and give it a specific data type
  + You may also allow the rule input to hold multiple values as an array
  + In Expression Mode of interface editing, rule inputs will be referred to in SAIL as your rule input’s name with the domain prefix “ri!”
    - A rule input named “employee” will be referenced as “ri!employee”
    - saveInto parameters in user input fields can be used to store data into rule inputs

* Associated Knowledge

**CDTs**

* Definition
  + initialism meaning “Custom Data Type”
  + a kind of Data Object that can be created in Appian Design
  + a developer-defined data structure.
  + analogous to Java “classes”
* Purpose
  + “Data Types allow designers to create a **logical grouping of related data** that can then be **used by other objects to share data**. Data can be shared internally, for instance between an interface and process model, or a web service that uses a CDT.”
* Implementation (Example)



* Alternative Rules
* Steps to implement
  + In your application within the Appian Designer main menu, click new and then data type
  + You will have multiple options in creating a new data type
    - Creating from scratch
    - Duplicating an existing data type
    - Create a data type from a table or view
    - Importing an XSD (XML Schema Definition)
* Associated Knowledge
  + [Data Types](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.z7d85qk8ngik)
  + XSD Files
  + CDT Best Practices

**CDT Best Practices**

Custom Data Type Relationships

* “Appian does not recommend configuring a field with a CDT type without an explicit relationship”
  + specify the columns that are or should become linked in the database table.
* Configured by default in CDT designer:
  + Many-to-one → a CDT is used for another CDT’s field’s type
  + One-to-many → if the array box is checked for a CDT-typed field
* Configure other relationships:
  + One-to-one → click the foreign key icon to further configure the relationship
  + Many-to-many → if it gets too complicated, download the XSD
* [Cascading](https://docs.appian.com/suite/help/20.4/CDT_Relationships.html#cascade) changes: When a CDT references another CDT, define the behavior for how those CDTs are updated when one is updated.
  + ALL (“master-detail”, “one-to-many”) → child has only one parent table
    - an Employee has an Address field, so if the Employee is deleted, so should the Address record
  + REFRESH (“reference” behavior) → related child tables are not updated when the parent is updated
    - a Patient has a Hospital field, but if the Patient is discharged, no change should come to the Hospital table.

Database Schema Best Practices...

* Make your CDT name and its fields’ names at most 27 characters long (for database purposes)
* (JPA annotations for @Table and @Column will use exact casing in the database entities based on your CDTs)
* Set nillable=“true” to avoid not-null constraints being imposed on your columns
* You can make a CDT based on a table/View from the database

Further notes on CDTs

* You cannot set the security level of a CDT. It inherits the security for the object that depends on the CDT (for example, an interface that uses a CDT determines the security of the CDT in that instance)

Associated Topics:

* [CDTs](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.xsjw2q4h65y)
* Data Store Entities

**XSD**

Definition:

* “**XML Schema Definition**” → Download the XSD to Edit an existing CDT
  + Must be a Designer or an Administrator
* [**XML**](https://www.w3schools.com/xml/xml_whatis.asp) (e**X**tensible **M**arkup **L**anguage) holds information, wrapped in tags.
  + It has no predefined tags; developers build software that can read the XML.
  + Software and hardware independent
* **JPA** (Java Persistence API) →  “The Java ORM standard for storing, accessing, and managing Java objects in a relational database.”
* Appian has a variety of [XML tags and JPA annotations](https://docs.appian.com/suite/help/20.4/Supported_XSD_Elements_and_JPA_Annotations.html) to know when designing/defining CDTs:
  + All tags start with “**xsd:**”

**Supported:**

|  |  |
| --- | --- |
| *XML element/JPA annotation* | *Purpose* |
| <xsd:schema> | wraps the rest of the schema, defines the namespace of the CDT |
| <xsd:complexType> | CDT name |
| <xsd:annotation>              <xsd:documentation> | annotation whose documentation tag includes first 1000 characters of the CDT’s description |
| <xsd:sequence> | the fields are defined in the order indicated by the sequence |
| <xsd:element> | defines a field. Has a name (keep under 27 characters for database compatibility), type, nillable attributes but also supports ref and form attributes |
| <xsd:annotation>      <xsd:appinfo source="appian.jpa"> | annotation whose source attribute indicates that JPA annotations can be used. Not all <xsd:element> will need an annotation  @Id → the element is the primary key, while  @GeneratedValue → there is a sequence to generate this value being used.  @Column(name="mName", length=40) → the column (name=”middleName” type=”xsd:string”) has a different name than the field/element itself and that the maximum characters for data is 40. |

Purpose:

* Can be used to **Create or Edit CDTs**
* **During CDT creation** with data type designer or database table**, the XSD is automatically generated.**

Implementation Example (what):

|  |
| --- |
| <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:tns="urn:com:appian:types:REM" targetNamespace="urn:com:appian:types:REM">     <xsd:complexType name="REM\_employee">         <xsd:annotation>             <xsd:documentation>                 <![CDATA[Read and write employee data from and to the Database]]>             </xsd:documentation>         </xsd:annotation>         <xsd:sequence>             <xsd:element name="empID" nillable="true" type="xsd:int">                 <xsd:annotation>                     <xsd:appinfo source="appian.jpa">@Id @GeneratedValue</xsd:appinfo>                 </xsd:annotation>             </xsd:element>             <xsd:element name="firstName" nillable="true" type="xsd:string">                 <xsd:annotation>                     <xsd:appinfo source="appian.jpa">@Column(length=40)</xsd:appinfo>                 </xsd:annotation>             </xsd:element>             <xsd:element name="middleName" nillable="true" type="xsd:string">                 <xsd:annotation>                     <xsd:appinfo source="appian.jpa">@Column(name="mName", length=40)</xsd:appinfo>                 </xsd:annotation>             </xsd:element>             <xsd:element name="lastName" nillable="true" type="xsd:string">                 <xsd:annotation>                     <xsd:appinfo source="appian.jpa">@Column(length=40)</xsd:appinfo>                 </xsd:annotation>             </xsd:element>             <xsd:element name="birthdate" nillable="true" type="xsd:date" />          </xsd:sequence>      </xsd:complexType>  </xsd:schema> |

Alternate Rules:

* You don’t usually need to mess with the XSD because the data type designer can help you rename and add fields. BY THE WAY: *Do not change the type of a field in the XSD*.
  + Instead, add a new field and create a new column in the database table.
* Map the field to the appropriate database column by adding the @Column(name= “column\_name”) to the xsd:annotation → xsd:appinfo tag if it already exists.
  + Otherwise, make a new annotation / appinfo set of tags between the element’s opening and closing tags, then add the JPA annotation between the appinfo tags.

Associated Topics:

[CDTs](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.xsjw2q4h65y)

**Appian Smart Services**

* Definition:

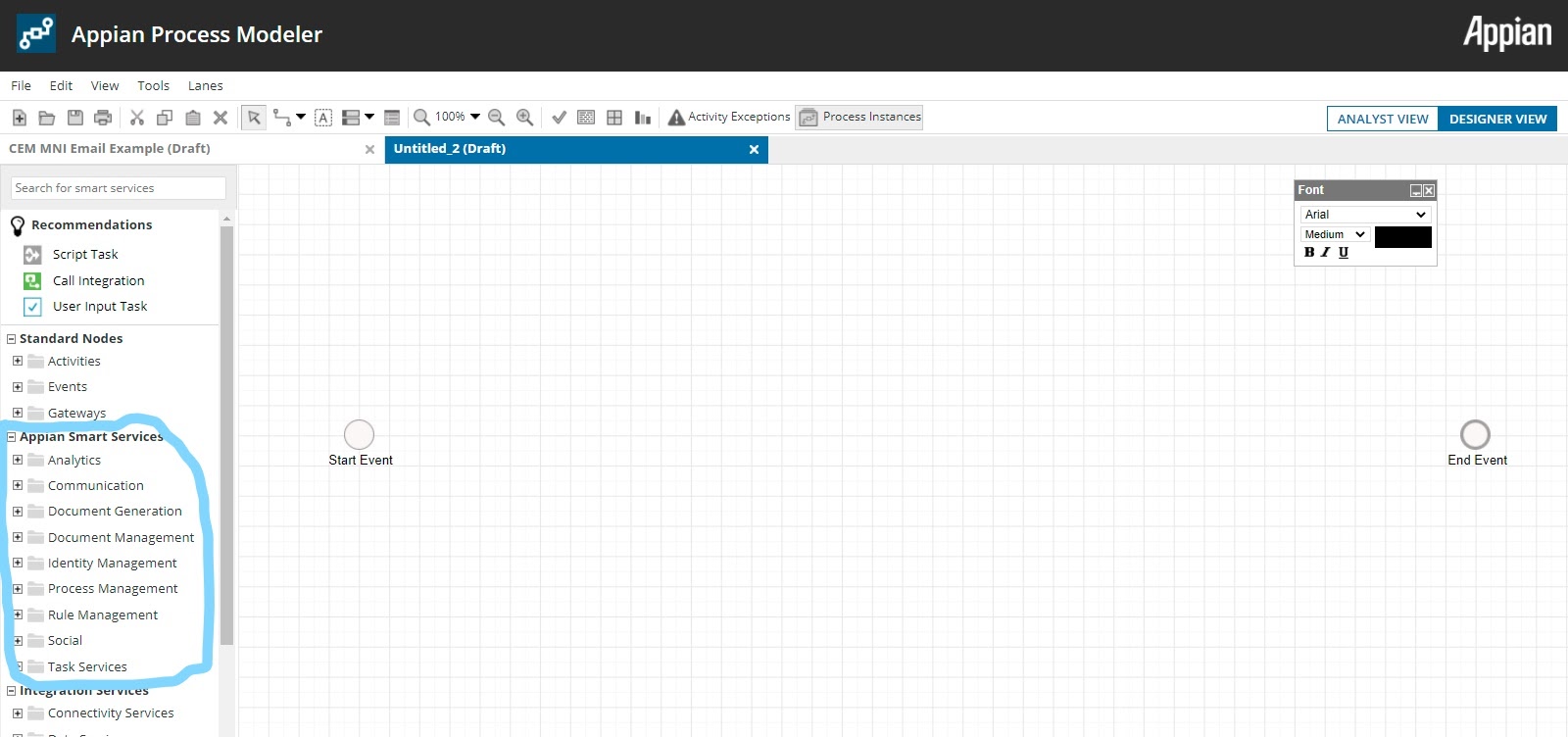
Smart services are flow activities that integrate specialized business services, like sending emails or writing data to a database.

* Purpose:

Smart services are available for use inside of Appian’s process modeler. These services can be connected to the flow of logic to perform actions necessary for a particular piece of business logic.

* Steps to Implement:

To implement any of Appian’s smart services we first need to be inside of the process modeler. We can do this by creating a new process model from the Appian Designer site. From the node selector on the left we can search through and select any of Appian’s smart services to drag and drop onto our process model.



* Alternative Rules:

Many of Appian’s smart services can also be called in any expression editor using function syntax. For example if you wish to create a new folder within an expression rule you could use the function:

**a!createFolder**(*name*, *parentFolder*, *parentKnowledgeCenter*, *onSuccess*, *onError*).

Write to da store from an interface

**a!writeToDataStoreEntity** (*dataStoreEntity, valueToStore, onSuccess, onErrorr*)

* Associated Knowledge:

Smart services include both standar smart services as well as Appian’s integration services. These integration services include both connectivity services as well as data service. A list of all smart services can be found here:

<https://docs.appian.com/suite/help/20.4/Smart_Services.html>.

**Rule Event**

*Definition:* Is an Event that can be added to a process model in two ways. First, can be added to the process flow as a process node, second they can be used within an activity to trigger an exception flow.

*Purpose:* With an expression, you can use a Rule Event to monitor global constants and process variables. Each time a global constant or process variable is changed, the rule is reevaluated

Steps to Implement:Configuring a Rule Event:

* In the Process Modeler, select and drag the Rule Event node from the tool palette to the canvas.
* Connect it to your process model.
* Double click the **Rule Event** node:
  + The Rule Event Dialog Box is displayed.
* Type a name for your event.
* Select the Setup tab
* Select **New Condition** or **New Expression.**

[**Rule Event - Appian 20.4**](https://docs.appian.com/suite/help/20.4/Rule_Event.html)

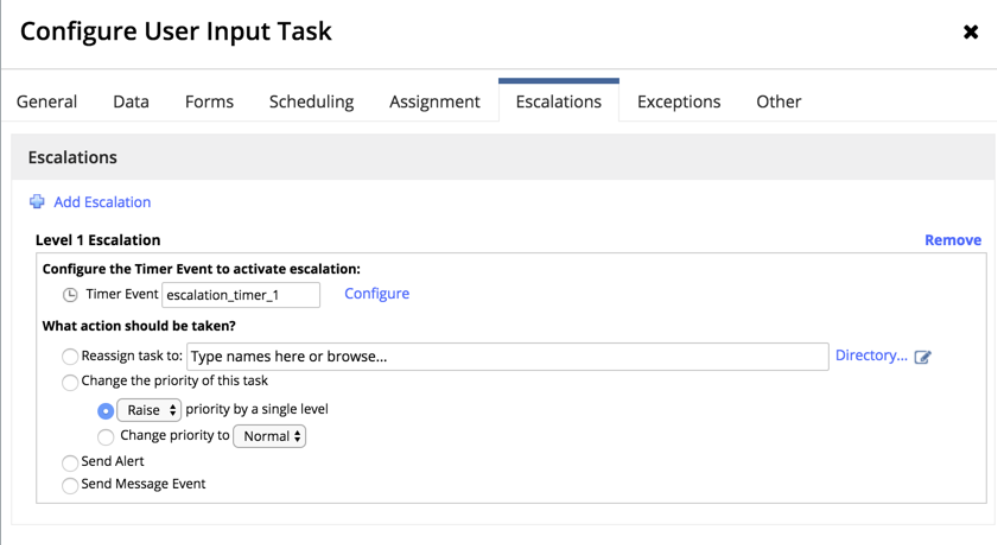
*Associated knowledge:* As a best practice, **recommend using conditions** rather than expressions whenever possible because conditions consume fewer processing resources and less memory than expressions

Escalation

Definition: A response that occurs when a task is delayed. There are two types of escalation: passive and active.

* **Passive escalation** - the process designer designates the amount of time during which a certain task must be completed by and the course of action that triggers in the event that it is not completed in time.
* **Active escalation** - enables a user with the appropriate permissions to escalate in the event that he/she feels that a task is not being completed quickly enough. Escalation actions may include reassignment to another assignee so the delay does not continue.

Purpose:

For *escalation tab:* If a task or process node is delayed for any reason, configurations in the escalations tab can automatically take a number of actions to solve the problem. An escalation can be configured for any attended node.

Steps to Implement:

The timer for the escalation is set by clicking the **Configure** link in the configure the timer event options. The *Timer Event* dialog box is displayed, which is identical in configuration to a [Timer Event - Appian 20.4](https://docs.appian.com/suite/help/20.4/Intermediate_Event_-_Timer.html).

Associated knowledge:

If an escalation is triggered, one of four actions can be triggered to occur:

* **Reassign the task to someone-** The task can be reassigned to any user and or group.
* **Change the priority of this task-** The task priority (tp!priority) can be raised or lowered.
* **Send Alert-** The activity will send an alert to a number of recipients.
  + When selected, an *Alert Message* dialog will pop-up, which will allow designers to configure a custom alert message.
* **Send Message Event-** The task can trigger a [Send Message Event](https://docs.appian.com/suite/help/20.4/Send_Message_Event.html) to another process.

**Interface editor**

**Definition:** The editor where interfaces are configured- there is a design mode with drag and drop functionality, and an expression mode where interfaces are configured using the SAIL language.

**Purpose:** A playground for creating interfaces, could also be used to design a quick

interface mockup for approval by management

**Key Features:**

* a place to add rule inputs(input variables for the interface),
* Edit tab (Design mode only)
  + Canvas- Middle section where interface objects are dragged and dropped
  + Templates section-on the right where common templates can be dragged onto the canvas
  + Palette section - A list of components to drag and drop
* Preview tab - A tab to see a preview of the interface(Where the interface is edited in expression mode)
* Performance tab- In this tab appian provides performance metrics for different expressions in the interface including  info on parameters, functions and expression rules
* A place where you can select the device type for previewing the interface on different sized screens.

**Implementation Example (how):** Not Applicable

**Alternate Rules:** Not Applicable

**Advantage/drawbacks:** Not Applicable

**Associated Topics:**

* Interface Components and Layouts,
* SAIL (Self-Assembling Interface Layer)
* ExpressionEditor
* Appian Design Mode

**Interface Layouts**

**Definition:**

* Layouts group similar components in an interface

**Purpose:**

* Provide visual alignment of contents and visual guidance to users in interacting with the interface.

**Implementation Example (what):**

* Top level layouts - formLayout (used for start and task forms), headerContentLayout()
* Each interface we have been building has one or  more layouts.

**Implementation Example (how):**

* **Header Layout**- provides a header that is ideal for landing pages and reports. This particular layout cannot be nested within other layouts
* **Section Layout** - Visually group related parts of interfaces
* **Card Layout** - Visually group related content- can change the background color
* **Box Layout**- A visually strong grouping of related components - Use box layouts sparingly as they can make pages look more cluttered.
* **Columns/Side by Side Layout** - specify horizontal arrangement of components across width of your pages. Side by side, allows for detailed managing of components.
* **Grids** - display tabular data in a structured, easy to scan layout. Keep concise and consistently formatted.
* **Billboard layout**-  shows content overlaid above a decorative background color, photo or video.

**Alternate Rules:**

**Advantage/drawbacks:**

**Associated Topics:**

* Interface components

**Interface components**

**Definition:** Interfaces are composed of different components

**Purpose:** To be used in interfaces to perform different functionality

[Documentation Page](https://docs.appian.com/suite/help/20.4/SAIL_Components.html#inputs)

**Implementation Example (what):**

See documentation link above for all components but here’s a couple examples and some categories

* **Input:**
  + Date - a!dateField()
  + Integer - a!integerField()
* **Selection:**
  + Radio Button - a!radioButtonField() - should use for short list of choices(less than 5)
  + Checkbox - a!checkBoxField()
  + Dropdown- a!dropDownField()- should use for moderately long list of choices
* **Pickers**
  + User Picker - a!pickerFieldUsers()- useful for long list (50 items or more)
* **Grids:**
  + Editable Grid- a!gridLayout()
  + Read-Only Grid - a!gridField()
* **Button:** a!buttonWidget() - There is only one
* **Link:**
  + Dynamic Link - a!dynamicLink() - Defines a link that triggers updates to one or more variables.
  + Record Link- a!recordLink() - Defines a link to a record view configured in the record type
* **Display**
  + Rich Text - a!richTextDisplayField() -Displays text in a variety of styles, including bold, italics, underline, links, headers, and numbered and bulleted lists. Also supports images and styled icons.

**Implementation Example (how):** Not Applicable

**Alternate Rules**:Not Applicable

**Advantage/drawbacks:**Not Applicable

**Associated Topics:**

Interface Layouts - these determine how the components are displayed, can nest layouts

SAIL- Components are a part of the SAIL framework

**SAIL (Self-assembling Interface Layer)**

**Definition:** Appian SAIL is the patented framework that takes declarative UI definitions to generate dynamic, interactive, and multi-platform user experiences. Definition directly from the docs, in my words it's just Appian's framework for creating interfaces.

**Purpose:** To create User Interfaces quickly that are cross platform

**Implementation Example (what):**

* Example SAIL functions are any Layout or Interface Component Functions(At least this is my understanding)
  + [Documentation Page](https://docs.appian.com/suite/help/20.4/SAIL_Components.html#inputs)

**Implementation Example (how):** Not Applicable

**Alternate Rules:** Not Applicable

**Advantage/drawbacks:**

**Appian docs(mostly marketing material and some of it is debatable)**

* Easy,
* Portable,
* Timeless,
* Powerful,
* Beautiful,
* Affordable

**Associated Topics:**

Interfaces components and layouts

**SAIL Recipes**

**Definition:** Recipes contain an expression that you can copy and paste into your interface in EXPRESSION MODE.

[Documentation Link](https://docs.appian.com/suite/help/20.4/SAIL_Recipes.html)

**Purpose:** Templates for commonly used functionality in applications

**Implementation Example (what):** [Create an Interface Wizard](https://docs.appian.com/suite/help/20.4/recipe-build-a-wizard-in-sail.html)

**Implementation Example (how):** Copy/paste the code into your own interface (expression mode)

**Advantage/drawbacks:**

Can quickly create functionality by using a template.

May take you some time to understand the template to use and adapt it to your needs

**Associated Topics:**

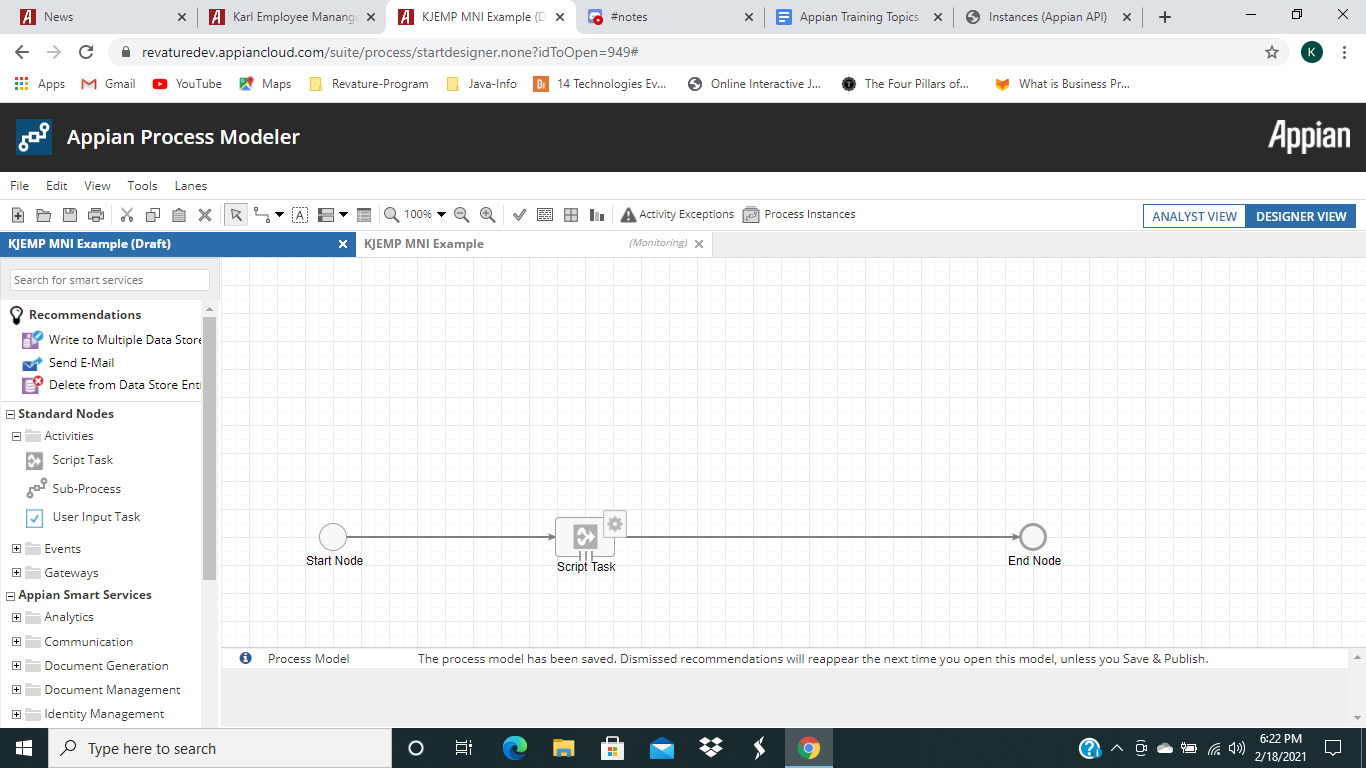
[Dynamic Interfaces](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.98v77zypvz9z)

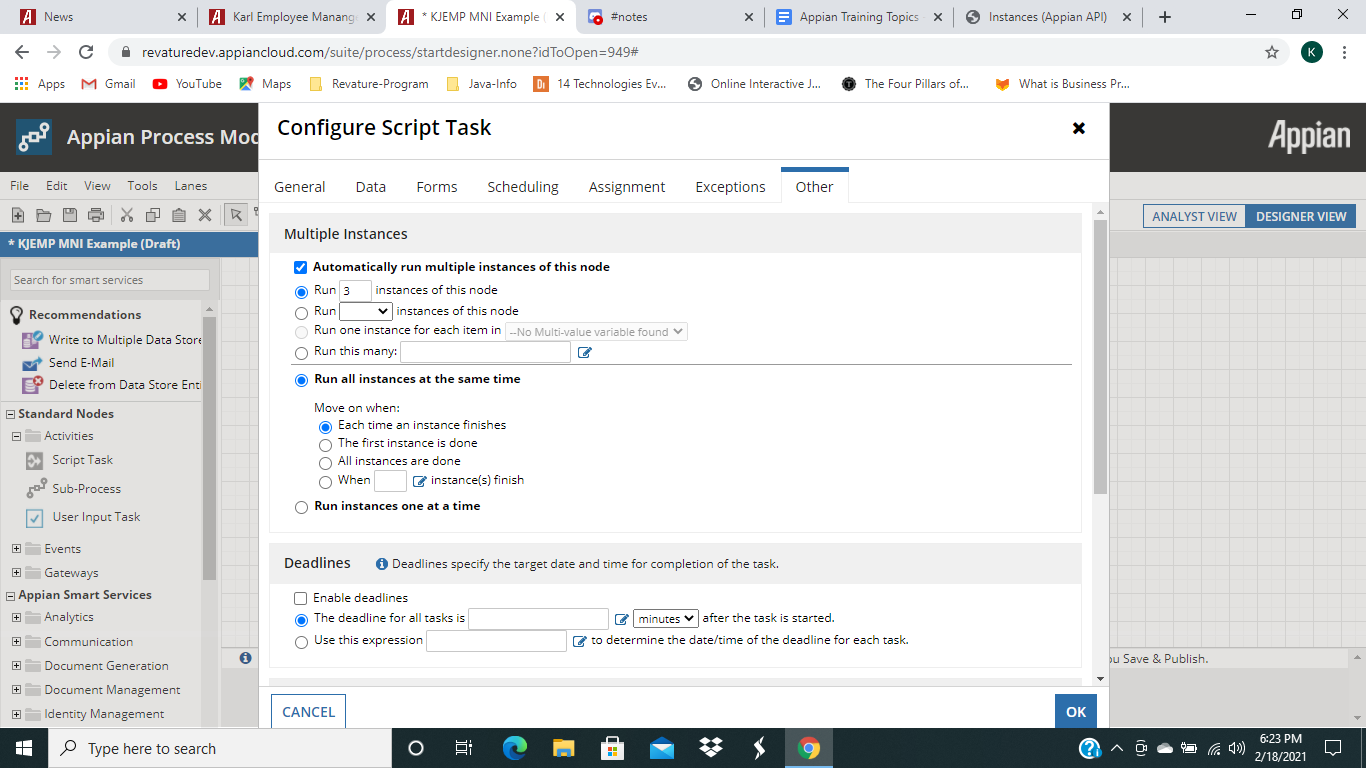
[Interface Editor](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.ysxzc87ek12b)

* **Multiple Node Instances(MNI)**
  + Definition:

It’s a process configuration that allows designers to start a process flow more than once by using the same flow token. It can be configured in the “**Other**” tab in the process model node properties.

* Purpose:
* To recreate the same task using a looping flow.
* To run the same task repeatedly on a schedule.
* To assign the same task to all members in a group.
* To assign a task for each item in a variable.
* To update an array.
* Implementation Example:





* Implementation Example (how):

To configure a node to run multiple instances:

1. Navigate to the other tab of the process model node
2. From there you would choose how many times and when the nodes will run the instance.
   1. Automatically run multiple instances of this node times let the designer choose how to run the instances of the node.
   2. Run all instances at the same time, with the option a developer has to choose when to move on after the nodes run through their instances.
   3. Run instances one at a time.

* Advantage/drawbacks:
  + Drawbacks: performance risk is higher when:
    - Processes containing MNI are not properly completed and deleted/archived.
    - The task containing MNI is part of an activity chain
      * Unattended nodes configured with MNI, at times, can cause poor user experience. If those nodes are configured to run multiple instances and also have activity chaining, they can cause performance issues as the wait times between chains might be longer. Moreover, the maximum 50 nodes capacity recommended is more likely to be exceeded through the activity chaining.
    - The MNI node is part of a loop.
* Associated Topics:
  + MNI Batch Limit
  + Health check report

**Records:**

**Documentation:** <https://docs.appian.com/suite/help/20.4/Records_Tutorial.html>

**Definition**:

* A record is an actionable view of data collected from various sources.
* It’s important for a record to be easily accessible, actionable, well-organized, and visually appealing. We’re able to configure records to only be accessible to those that need to see it, through security.

**Purpose**:

* We use records to conveniently view and edit our application data, all at once, regardless of where it’s contained within the application.
* allow you to bring together your internal and external data sources and see your business information from a number of different views and perspectives. This may include searchable record lists, read-only grids, summary views, charts, and reports.

**Implementation Details**:

Many steps are **OPTIONAL**

**Steps:**

1. **Create Record Type**: Do this like we do any other object. The name is what designers see in the Appian Designer, and the Plural Name is what users see when accessing the records. Set the security as needed for your application.

* **Choose your data source**. For our employee record, we used “Database” and selected our employee DSE from our data store.

1. **Configure Record List**:

* **Configure record list columns**: for example, we changed our employee ID column to be a link to the summary view instead of our first name column
* **Create constants**
* **Add a user filter**

1. **Create Record Views and Actions**

* **Create a summary view interface**: i.e. our employee interfaces we made a long time ago
* **Add summary view to record type**
* **Configure record header background and title**
* **Create an update employee interface for a related action**: again, we used our old employee, employee secondary, employee address, employee merged interfaces here
* **Create a process model for a related action**: We created new process models for our related actions. Logically, they were very similar to our old process model, but the big difference is that the process variables are parameterized and we need to enable activity chaining between the start node and the initial form.
* **Add related action to record**

**Implementation Example:** Refer to the Employee Record we created during class.

**Related Topics**:

* **Record-centric approach/design**: designing your application processes around taking action on the data that’s displayed in a record (plan your application with records in mind).
* **Tempo, List, Views, Default Filters, User Filters, Related Actions, etc**

**Default Filters:**

**Documentation:** <https://docs.appian.com/suite/help/20.4/Record_Type_Object.html>

**Definition**: Default filters automatically filter out data *before* it’s displayed to a business user. They can be static (field) or expression based.

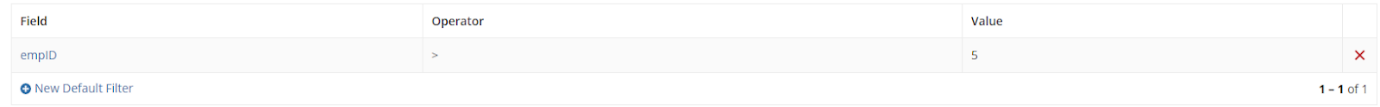
**Purpose**: A default filter displays a subset of records to the user from the record source based on filter criteria. When a record is filtered out by a default filter, it does not show up in the record list, is not returned in queries on that record type, and users may not access its views. Controlled by the developer (users don’t have any control over default filters).

**Implementation Details:**

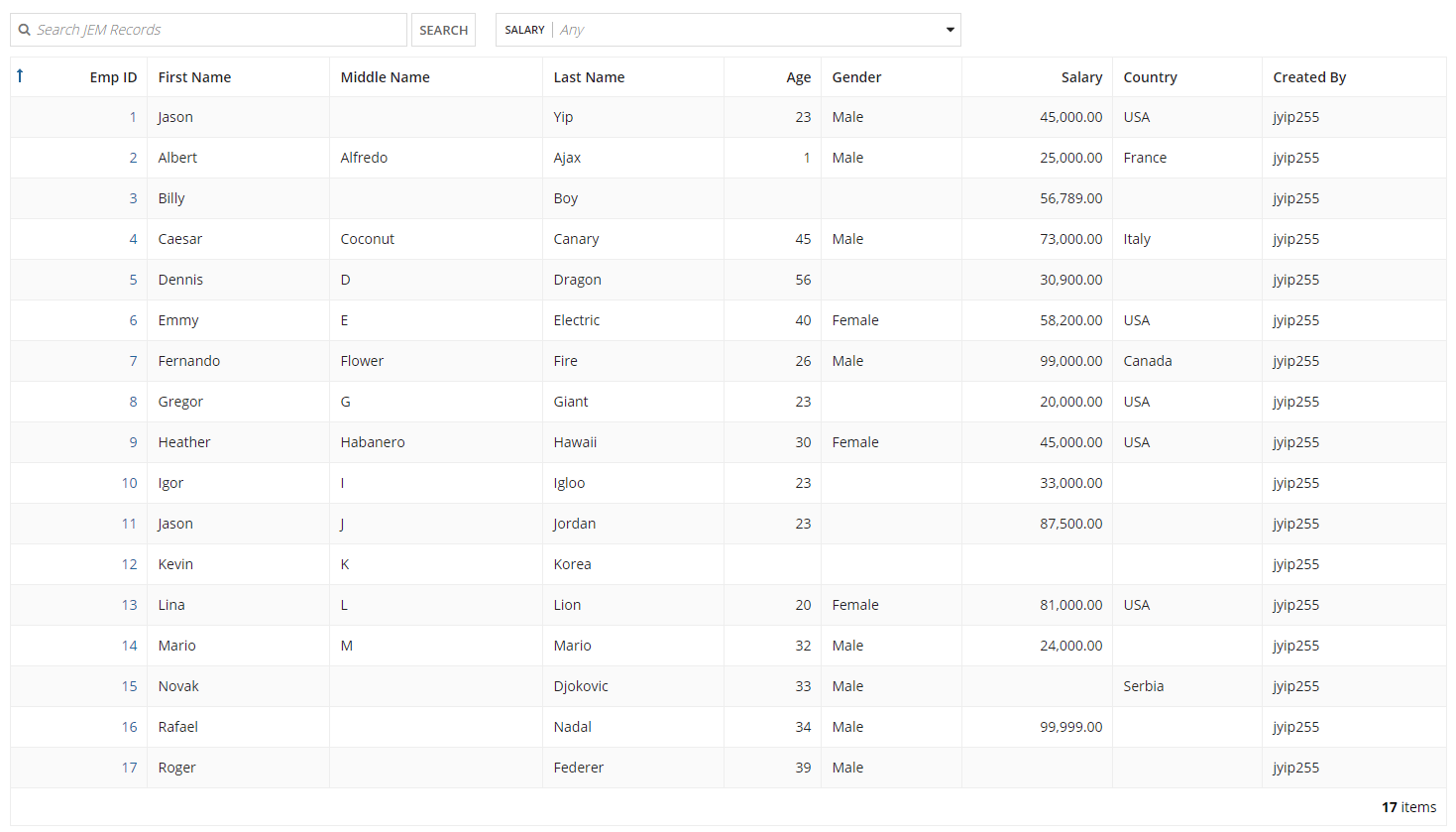
* Located in the **“Source & Default Filters”** tab of a record.
* **By Field:** Developers can apply multiple default filters to a record type. All filters are joined by an AND union.
* **Expression**: Developers can also specify more complex default filters on the record type using an expression containing a list of a!queryFilter() or a!queryLogicalExpression().

**Implementation Example**:

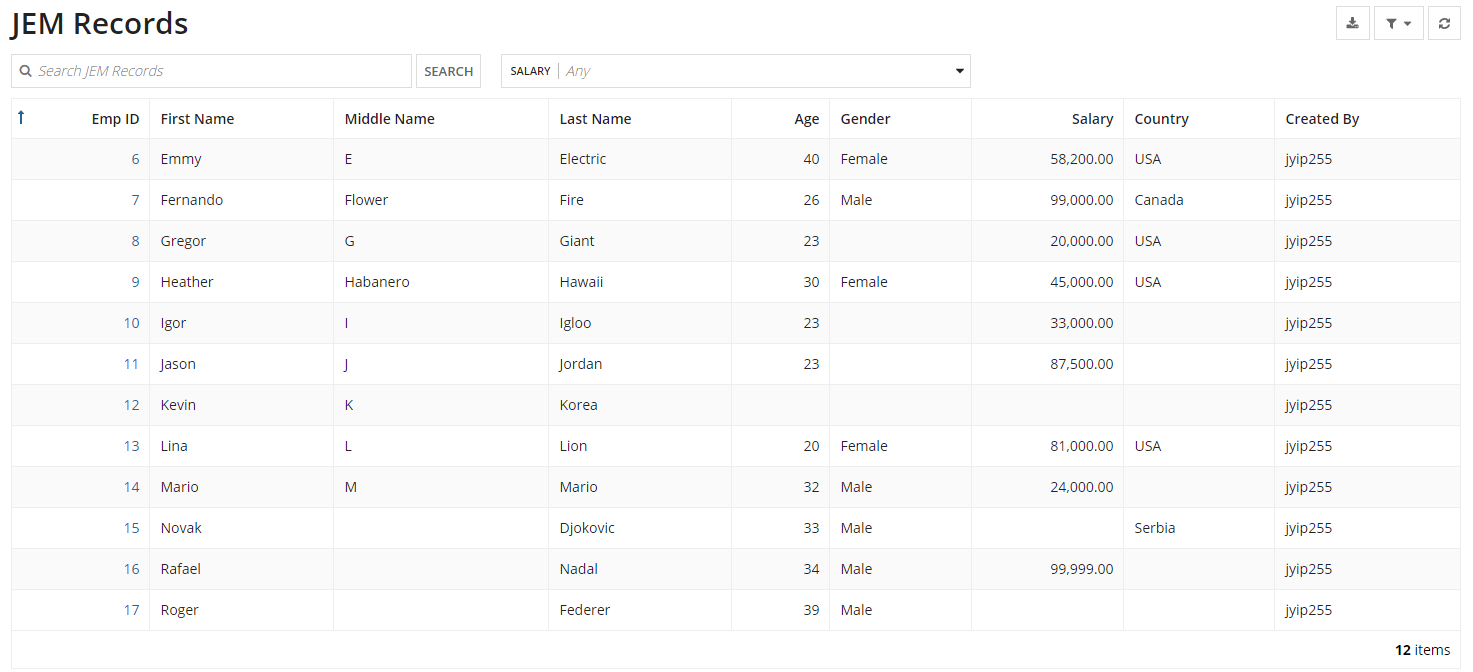
Filter (empID > 5):



Record without filter:



Record with filter:



**Related Actions**

* Definition:
  + Documentation link:<https://docs.appian.com/suite/help/20.4/Using_the_Records_Tab.html>
  + Related Actions is one of the different tabs in a record that allows for initiating actions related to the current/specific record.

* Purpose:
  + The main purpose for it is to conveniently utilize functions on the records that you had fetched from the data store

* Implementation Example (what):
  + In terms of our Employee Management System, an example of a related action would be to update an employee entry in the record. Similarly, you can edit any other entity as long as you utilize the appropriate data store entity.

* Implementation Example (how):
  + Once the record has been created, to the left there’s a menu that contains the more important parts of a record, being source and default filters, tempo, list, user filters, views, related actions, and performance.
  + Based on how we went about making our record, we made the related actions after making the views and filters that we wanted to implement, which makes sense due to wanting to see all of the data that you plan on manipulating.
  + When you do go to the related actions menu and press new related action, it asks you for the specific process model you want to base the action on, context which is an expression that declares variables for the process, and a name, key, and optional description field. You can also change the icon.
  + Once created, it should appear next to the user views when you saw a specific record’s entry.

* Alternate Rules:
  + Not exactly sure how this would apply to related actions, generally it depends in terms of what kind of related action you want to perform on the record but it’s always going to be the same setup with some variance in creating the related action with either using the process model name, entering the information yourself, or plainly using an expression.

* Advantage/drawbacks:
  + The biggest advantage is how they can be implemented to manipulate the data store that the record is connected to.
  + Since the record can be made public to the tempo, this means that these related actions can be displayed to the end user for convenient manipulation of data.
  + The only downside I can think of would just be that you have to set it up and it can be a bit confusing in terms of all the different nuances to making it.

* Associated Topics:
  + The main associated topics would be a record and all the different aspects of a record, such as the list view, setting up filters, and views.

**Entity-Backed Records**

·       Definition:

o   Documentation link:<https://docs.appian.com/suite/help/20.4/Using_the_Records_Tab.html>

o   Basically there are 4 different ways to create a record, this particular way is when you create a record based on a data store entity.

·   Purpose:

o   The main purpose for it is one of several different ways of creating a record. If the designer wants their record to be directly connected to the data store in terms of accessing information and making transactions with that data store then entity-backed records are the best option for your business.

·   Implementation Example (what):

o   In terms of our Employee Management System, an example would be the record that we created. In specifying our data store, we were able to pull a list of all of the saved records inside of the data store, create search filter options, a summary page as well as a more in-depth page discussing all the details of the employee, and a related action of updating their information in the data store.

·   Implementation Example (how):

o   Once the record has been created, to the left there’s a menu that contains the more important parts of a record, being source and default filters, tempo, list, user filters, views, related actions, and performance.

o   Based on how we went about making our record, we made the related actions after making the views and filters that we wanted to implement, which makes sense due to wanting to see all of the data that you plan on manipulating.

o   When you do go to the related actions menu and press new related action, it asks you for the specific process model you want to base the action on, context which is an expression that declares variables for the process, and a name, key, and optional description field. You can also change the icon.

o   Once created, it should appear next to the user views when you saw a specific record’s entry.

 ·   Alternate Rules:

o   In making a record, there are a total of 4 different options when you’re setting up the record. Based on what you want for that record’s functionality, you set the source type to a database, a process, Salesforce, or some other web service. Each have their own different implications for what you want to accomplish in your project.

·   Advantage/drawbacks:

o   The main advantage of an entity backed record is as stated earlier, there’s access to the data store directly that you created in Appian.

o   The main drawback is that if you want connectivity outside of Appian, you’d have to select a different option, such as Salesforce or other.

·   Associated Topics:

o   The associated topics for this would be to study the other different types of records, as well as general information on what records are.

**User Filters:**

**Documentation:** <https://docs.appian.com/suite/help/20.4/Record_Type_Object.html>

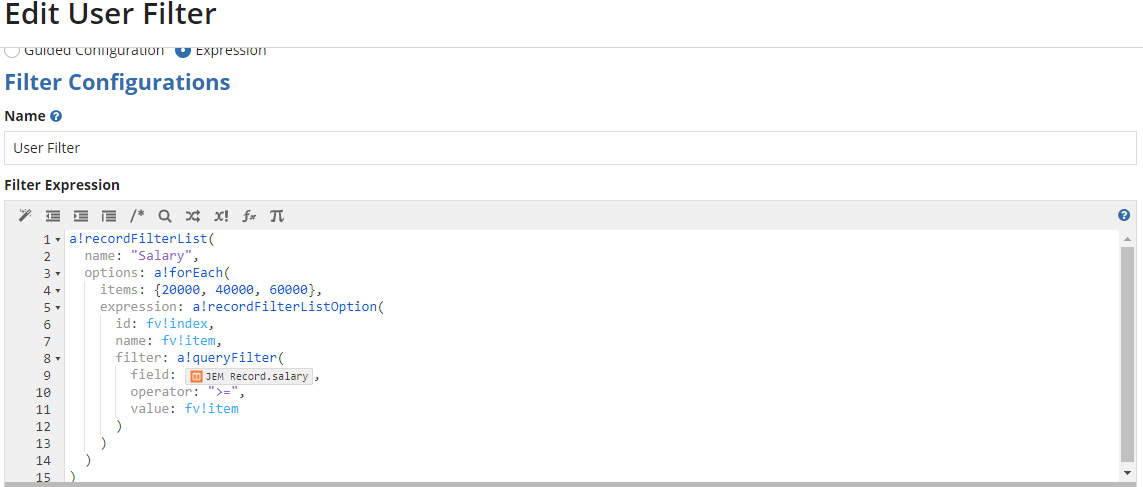
**Definition**: User filters offer the ability to filter subsets of record data by selecting options in a list or using a date range style filter.

**Purpose**: Functionally very similar to default filters. Except instead of filtering out data before it’s shown on the record, users can use these filters to filter through information on the record itself. These filters are what the *User* interacts with while looking at a record.

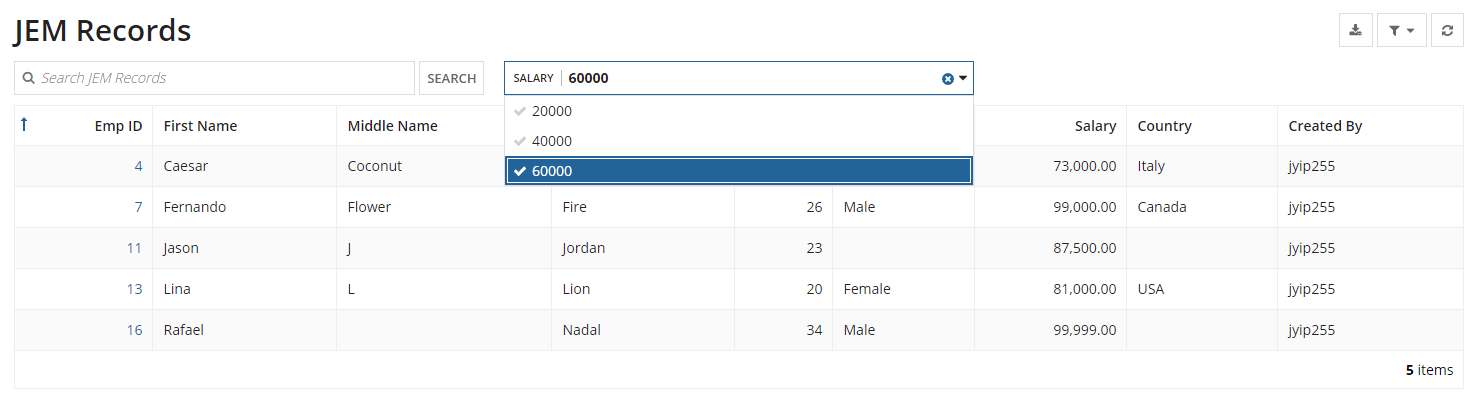
**Implementation details:**

* Located in the **“User Filters”** tab of a record.

**Implementation Example**:

****

**Show all records where salary > 60000:**

****

**Database Smart Services**

* Definition:

There are four smart services for interacting with data and databases. They are **Query Database**, **Delete from Data Store Entities**, **Write to Data Store Entity**, and **Write to Multiple Data Store Entities**.

* Purpose:

The purpose of database smart services is to allow our process models to interact with databases and data stores.

* Implementation:

1. To implement a **Query Database** smart service drag the service onto your process model. Then open the properties and select the “Setup” tab. This tab requires two configurations. The first is the database connection. We can either use a pre-existing data source within our application or we can configure a custom data-source. The second configuration is the queries. Here we can create multiple queries. Once your queries are configured you can find the results in the output section of the “Data” tab.
2. To implement a **Delete from Data Store Entity** smart service again begin by dragging the service onto your process model. The primary configuration is the node input “Data to Delete”. This specifies which data entity and the primary key for the items to be deleted. Here is some example syntax:

={

{entity:pv!ENTITY\_SPRINT\_ITEMS, identifiers:pv!itemIdsToDelete},

{entity:pv!ENTITY\_COMMIT, identifiers:pv!commitIdsToDelete},

{entity:pv!ENTITY\_SPRINT, identifiers:pv!sprintIdsToDelete}

}

1. To implement a **Write to Data Store Entit**y smart service drag the service onto your process model. In the “Data” tab you must configure the node input “Data Store Entity” with the entity you wish to write to. It is common practice to utilize a constant value for this. Next you need to add a custom input with the value(s) you wish to enter. You may write multiple values with this service but only to a single entity.
2. To implement a **Write to Multiple Data Store Entities** smart service drag the service onto your process model. In the “Data” tab you can configure the “Value to Store” input variable with the data store entities and the data values to write. Here is some example syntax:

={

{entity:pv!ENTITY\_OPPORTUNITIES,

 data:{pv!RadiationOpp,pv!NewBusinessOpp}},

{entity:pv!ENTITY\_ACCOUNT,

 data:{pv!AcmeAccount}},

{entity:pv!ENTITY\_AE\_NOTES,

 data:{pv!ListOfNotesFromAccountExec}}

}

* Alternate Rules:

1. These smart services are treated as single transactions. This is especially important to note for the Delete from Data Store Entity and the Write to Multiple Data Store Entities.
2. In the Query Database smart service no \* selections are allowed, only 1 select query is allowed, and you should not insert “;” at the end of your queries.
3. The Delete from Data Store Entity, Write to Data Store Entity, and Write to Multiple Data Store Entities smart services can all be called as functions from an expression.

**Query Entities**

* Definition

A query entity refers to the function a!queryEntity().

* Purpose:

The a!queryEntity() function is used to execute a query on a specified **data store entity** and return a data set.

* Implementation:

The a!queryEntity() function has the following basic syntax.

**a!queryEntity**( *entity*, *query*, *fetchTotalCount* )

1. entity - this parameter refers to which data store entity we are making a query through. This value should be passed as an application constant.
2. query - the query parameter must be entered using an a!query() function. This function has the following syntax.
   * **a!query**( *selection*, *aggregation*, *logicalExpression*, *filter*, *pagingInfo* ).
     1. selection: (Optional) A list of fields you wish to retrieve. Each element is created with an a!querySelection() function.
     2. Aggregation: (Optional) A list of fields and aggregating options you wish to perform. Created with a!queryAggregation().
     3. logicalExpression: (Optional) A set of operations to apply to the data before any grouping or aggregation. Created using a!queryLogicalExpression()
     4. filter: (Optional) a single condition to apply to the data before any grouping or aggregation. Created using a!queryFilter().
     5. pagingInfo: The paging and sorting configurations to apply to the data.
3. fetchTotalCount - If set to false, avoids running the query that retrieves the total number of rows in the totalCount parameter of the resulting datasubset. In cases where you don't need the total count, skipping this query can improve the performance of a!queryEntity(). Default: false

* Alternate Rules:
  + If called multiple times within a single expression to the same parameters only one query is made.

[**Expression Rules**](https://docs.appian.com/suite/help/20.4/Expression_Rules.html)

**Definition**:

* An expression rule is a stored expression that can be called from other expressions.
* A designer built function (vs appian predefined functions)
* Like Appian functions, expression rules always return a value that may be influenced by one or more inputs.
* Expression rules differ in that their return value is dictated by an expression.
* Expression rules can be called from any expression, so they can be reused across multiple objects throughout the system.

**Purpose**:

**Implementation Example (what):**

* Test any function in Expression Rule object
* Used expressions to query from database, example: fetch all Employees from the db table

**Implementation Example (how):**

* On Designer Mode → New → Expression Rule → (Two options, Create from scratch & Duplicate existing) → Give Name, Description and Save in folder, Rule inputs (used to pass data into the expression rule).
* When the expression page is open → use [**a!queryEntity()**](https://docs.google.com/document/d/1cWCSIFDgzBR0xpMaevrRrodOEW6IO7aqm1ZtYvu7N0Y/edit#heading=h.oetikyv98phe)function

**Alternate Rules:**

Advantage/drawbacks:

Associated Topics:

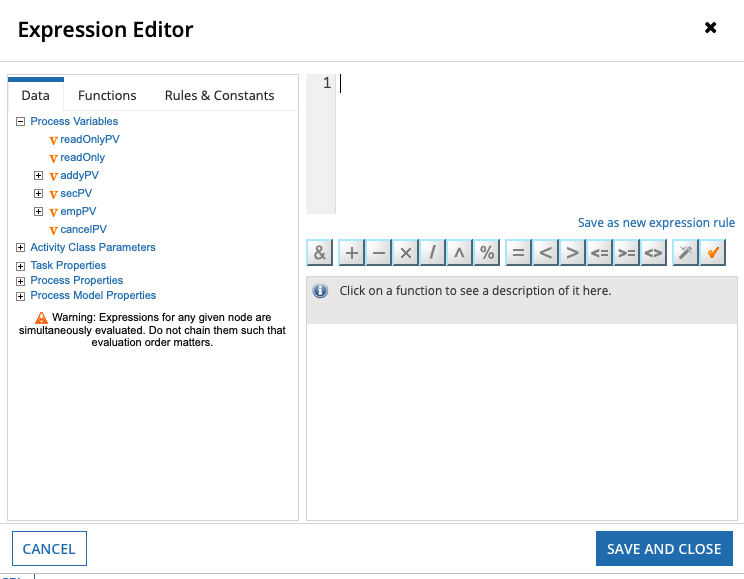
**Appian Designer Mode:Interface**

* Definition:
  + One of the two ways to make an interface in Appian
  + Allows you to drag and drop components instead of typing the code
* Purpose
  + To make interface design quick and easy
* Implementation Example
  + Open up any ui and you have your example of design mode
* Alternate Rules
  + Besides giving you drag and drop functionality design mode also gives you templates to start from
    - Templates range from just some form components on the page to actual examples for forms like a request form or an orders dashboard
  + Design mode also gives you builders
    - This is where you can create an interface from a CDT
    - There is also a report builder
      * This allows you to pick a constant and you can pick different ways to show the report.  Read only grid, pie chart, bar chart, column chart ,a line graph
* Steps to Implement
  + When you create or open a UI it default opens to the design mode
* Associated Knowledge
  + Certain layouts cannot be included in other layouts
    - A form layout has to be the outermost layout
    - A section layout cannot include another section layout
    - A side by side layout cannot be included in itself
  + Anything you do in design mode will be reflected in Expression mode as well
    - Several things we’ve been doing in Expression mode (label,showWhen,contents) can be edited in designer mode on the right hand side of the screen

**Dynamic Interfaces**

* Definition
  + A ui that changes based on user input or data it reads in.
* Purpose
  + To tailor an interface to the user to make it easier to use
* Implementation Example
  + Using showWhen() is a good way to create a dynamic interface.  You hide the information and fields until the user needs them so they don’t get intimidated or confused with all the clutter
  + Another one would be the report we created on 2/18/2021.  The interface changes depending on the data and the selection you make.  It also includes record links to each record you can view.
* Alternate Rules
  + Not everything has to be as fancy as using a showWhen() or dynamic link.  Something as simple as changing a label using an if statement based off of input is considered as dynamic
* Steps to Implement
  + Decide what components will change
  + Find the appropriate function to use to achieve your goal
  + Use that function
* Associated Knowledge
  + There is no real documentation on dynamic interfaces and even just the interfaces documentation doesn’t really mention it.  This seems like a big picture topic as opposed to some other topics

**Expression Editor**

* Definition
  + The UI within Appian to create or edit expressions
* Purpose
  + To enable the designer the capabilities of creating or modifying a query, property, or object to retrieve or manipulate data within an Appian Object
* Implementation example
* 
* Alternate rules
  + Appian Smart services are more user friendly and structured ways of modifying the data that is used within an object, especially in Project models or interfaces.
* Associated knowledge
  + Expressions are vital for the functioning of data being transferred between objects.
  + Making sure that rules that are properly configured and syntax is persistence will prevent some errors from occurring.
  + Expression mode can allow for larger scale expression editor within the interface objects
* Steps to Implement in 5-6 steps
  + Within a process model, and a process node to the model
  + Right-click on the process node and enter the properties
  + Go into the data section of properties
  + Click on a property and next to the value parameter, click on the pen to enter expression mode

**Security**

* Definition
  + Keeping access to objects limited to only users who need them
* Purpose
  + To keep users from having access to everything, causing confusion and opening the door to further issues
* Implementation Example
  + You have an object that only the HR team needs access to.  Add security to it so HR has viewing permissions and all other users don’t have any permissions on the object.
* Alternate Rules
  + You can add groups instead of individual users
    - Using groups allows you to add members dynamically also by using membership rules
      * A membership rule would let you say “Anybody in a group with “associate” in the name gets added to this group” which is then assigned an object
* Steps to Implement.
  + Adding security to an object
    - Usually when you create an object it will ask you for security permissions and you can grant users permissions there.
    - Select the user and level of permission (Admin, editor, viewer)
    - If you don’t do this on creation you can do it later as well
      * Select your object
      * Across the top of the designer select the new “Security” option
      * Follow the steps like you would if you were doing it during creation
* Associated Knowledge
  + It is best practice that pretty much everything should have a group or user with Admin privileges and one with Viewer permissions
    - Some objects require these and more, like an Initiator privilege.
      * Adding a process model as an application action is one that requires the model to have someone with initiator permissions

**Appian Functions**

* Definition
  + Prebuilt expressions in Appian that allow a designer to do a myriad of stuff
* Purpose
  + To make the creation of applications fast and easy by not having to make all these functions ourselves
* Implementation Example
  + Any time an a! has been used is an example of an Appian function being implemented
  + You start to make an interface and want to start with local variables.  You use the function a!localVariables() which takes any number of local variables and one expression.  For the expression you start by using a!formLayout().  Inside that you make a box layout using a!boxLayout().  And inside the box you give some inputs using a!textField and a!integerField
* ·         Alternate Rules
  + The documentation states the fn! Prefix is also for some functions, usually when using a function inside a function.  Examples of this we would know would be fn!isnull, or fn!if
    - An example of this is use is with the reject() function.  If you had a list and you wanted to reject all the null values the syntax would look like “reject(fn!isnull,ri!listname)
* ·         Steps to Implement.
  + Appian takes care of making the syntax look good if you know what function to use.  Just type in “col” and select “ColumnLayout” from the list.  It will auto format to a!columnLayout for you.
* ·         Associated Knowledge
  + Im just link the page with all the functions so you can explore each [function](https://docs.appian.com/suite/help/20.4/Appian_Functions.html) for yourself

**Editable Grid**

o   Definition

§  A grid that displays data and allows for the values within the cells to be changed

§  Arranges interface components in a tabular layout to provide quick inline editing of fields.

o   Purpose

§  To allow for quick editing of records in a grid view in a UI

o   Implementation Example

§  The work we did in class. Added an editable grid to a user interface and persisted changes made to that grid to the database

o   Steps to Implement

§  Use the gridLayout() function

§  Add the necessary parameters in the grid

o   label & instructions lie above the table

o   headerCells are read only cells containing labels for each column

§  a!gridLayoutHeaderCell()

o   rows contain the layouts for each row, the number of row cells must match the number of header cells

§  a!gridRowLayout(

·       contents:{

·       a!textField/a!integerField/a!floatingPointField/etc.})

§  ^configures one row, each gridRowLayout configures a row

·       Or can be configured using a foreach loop of all items in the list

o   Only difference is gridRowLayouts are only declared once within the foreach loop

o   Associated Knowledge

§  In order for changes to persist to the database there needs to be a write to data store entity function written into the expression somehow, just editing the values in the cells will not make changes to the database

**Read-Only Grid**

o   Definition

§  A grid that displays data, just to be displayed

§  Arranges interface components in a tabular layout to provide quick view of fields.

o   Purpose

§  To display data in a grid view in a UI

o   Implementation Example

§  The example worked on in class, displaying all of our employees in a read only grid

o   Alternate Rules

§  This grid does not necessarily mean that you cannot edit the values within it, the selected row can still be expanded on outside of the grid in another layout.

o   Steps to Implement

§  Use the gridField() function

§  Add the necessary parameters in the grid

o   label lies above the table

§  a!gridColumn contains the label for the column, along with the value for each row, using the fv!row variable to access different fields of the cdt in the list

·       Or can be more easily configured in design mode just by clicking read only grid and choosing the data source for the table

o   Associated Knowledge

§  The read-only grid provides benefits over the paging grid in these ways

§  Better UX (column widths, header tooltips, rich text).

§  Native handling of common data errors when you use fv!pagingInfo.

§  Deterministic sorting (from secondarySorts).

§  Helpful function variables for passing selected row data (fv!selectedRows, fv!deselectedRows) that can be used to avoid re-querying data

**Cascading Dropdown**

o   Definition

§  A user selection that presents a placeholder then all of your selection options when you click on it

o   Purpose

§  To save space when it comes to making a from a very long list of options

o   Implementation Example

§  A cascading dropdown selection of US states

o   Steps to Implement

§  Use the dropdownField() function

§  Add the necessary parameters in the grid

o   label lies above the table

o   placeholder is displayed in the selection before it is clicked on

§  choiceLabels display the labels of the choices

§  choiceValues are the corresponding values for the choice labels, the number of choice labels must match the number of choice values

§  saveInto to save the choice value into a target

**Sites:**

**Documentation:** <https://docs.appian.com/suite/help/20.4/Sites.html>

Definition:

Purpose:

Implementation Example (what):

Implementation Example (how):

Alternate Rules:

Advantage/drawbacks:

Associated Topics:

**Start Form:**

Definition:

* a start node configured with an interface
* No need of activity chaining because form is populated when the PM is invoked

Purpose:

Implementation Example (what):

Implementation Example (how):

Alternate Rules:

Advantage/drawbacks:

Associated Topics:

**Exception:**

Definition:

* Unresolved user input task

Purpose:

Implementation Example (what):

Implementation Example (how):

Alternate Rules:

Advantage/drawbacks:

Associated Topics:

**Agile/Scrum**

**Definition**: Agile is a series of software development methods that focus on iterative design and adaptation to changing requirements for a project. Scrum is a framework for Agile, and is much more commonly used in software development.

**Purpose**: As previously mentioned, Agile and Scrum development are used primarily to ensure a project can adapt to goals that can potentially change at any time. It has also been known to be significantly less stressful on developers than more traditional development cycles, as well as allowing for extra flexibility on deadlines should unforeseen obstacles occur.

**Implementation Example**: While Agile itself is more of an overall guideline rather than something that can be strictly adhered to, Scrum is much easier to use in projects. In a Scrum-based project, several components will be required in order to keep with Scrum methods. First would be the backlog, where all the goals of the project will be listed, then will be the sprint backlog, which will have all tasks that will be handled during the current sprint cycle. Then you have the sprint, which consists of another component: the daily stand-up, which is a short 15 minute meeting updating progress and explaining obstacles. A sprint cycle is usually somewhere between two weeks and a month long, but is very rarely longer than that.

**Advantage/drawbacks**: Scrum and Agile are very good at adapting to changes in a project’s lifestyle or in a project where changes in technology is expected. However, the process can be convoluted at times and some parts of Scrum will slow down the development process in comparison to more traditional methods when a project’s requirements are static.

**Data Store Objects**

**Definition**: Data stores are objects that hold references to external databases

**Purpose**: Data stores are useful for not only for storing information, but are useful for accessing and modifying database tables because they allow the manipulation of databases without having to use SQL code.

**Implementation Example**: To create a new data store, all you have to do is click on the ‘new’ drop down menu and then select a data store, and finally name your new data store. The only other part of creating a data store is attaching data entities.

**Advantage/drawbacks**: The main advantage of Data Store objects is the fact that they abstract away normal SQL commands and allow for the more intuitive Appian language to be used to access databases instead. The drawback is that data stores can only grab data that is 1MB or smaller with any given data grab.

**Associated Topics**: Data store entities are closely tied to Data Store Objects, since Data Store Objects store Data Store Entities.

**Data Store Entities**

**Definition**: Data Store Entities are the individual references to different database tables

**Purpose**: To link a Custom Data Type to a database table

**Implementation Example**: In order to create a Data Store Entity, start with an already existing Custom Data Type. Go into your Data Store Object and add an entity. This will prompt you to select a type and give it a name. Generally we want the name to be the same or similar to the name of the custom data type. After you have selected both the name and the entity, click ‘ok’, and then scroll down to click ‘verify’. Finally, make sure to have the box ticked that automatically generates the table based off the Custom Data Type, then hit ‘Save % Publish’

**Associated Topics**: See Data Store Objects

**Task report**

**Definition:**

**Purpose:**

* Task reports display task information with a link for users to open each task and begin working on it.
* When you save a Tempo report as a task report, it appears on the Tasks tab. Task reports are also used with sites.

**Implementation Example (what):**

* Configuring a task report on Site

**Implementation Example (how):**

1. Create **constant**
   * Name:
   * Description:
   * Type: **document**
   * Value: **active\_tasks** (list of tasks created by Appian in the back end)
2. Create interface
   * May need to use a!localVariables() function and local variable, so that we can see what the return of a!queryProcessAnalytics() function is, as well as to see the fields in taskReport.
   * Use **GridField**
     + Parameters **- data** and **columns**
     + In the **data** parameter, use **a!queryProcessAnalytics()** function; this function is used exclusively for querying task reports only.
       - Two parameters of the function,
         1. **report**: the constant created on step 1 (type: document, value: active\_tasks)
         2. **query**: created with a!query()
     + In **columns** parameter, use an array of columns created by a!gridColumn() function and value for each column can be found with dot notation from fv!row.
       - Example: fv!row.c0 - name of the task, fv!row.c2 - created by, fv!row.c8 - assignee/s
3. To link each task row from the grid to the actual task, use **a!linkField()** function in a column value,
   * Parameters of the a!linkField() function
     + links: created by using function **a!processTaskLink(label: fv!row.c0, task: fv!identifier)**
       - Task is the unique identifier for a specific task,

Alternate Rules:

Advantage/drawbacks:

Associated Topics:

**Tempo**

Definition:

* Appian’s Tempo is basically a UI framework that unites all the different parts of the business app that you’re developing.
* <https://community.appian.com/learn/b/feature-highlights/posts/appian-tempo>
* The main parts of the tempo are the news, tasks, records, reports, and actions.

Purpose:

 The main purpose for it is to be essentially the front-end part of your business application, so that end users or clients can utilize your functions, database, etc.

Implementation Example (what):

o   In terms of our Employee Management System, we need our Tempo to contain the functions that we intend a user to complete. So, after publishing our record along with a fully featured process model that adds an employee, we’d be able to allow users to apply as employees, then a manager would decide if they wanted to accept or reject them. Or a manager could go through a report on how many employees were accepted or rejected. The news tab keeps everyone connected, the tasks are for employees that have a certain thing assigned to them to complete, such as needing to fill out a form, a record can display all of the information in the database along with related actions, a report can dynamically present the database, and actions are generalized actions not directly related to a specific record.

·   Implementation Example (how):

o   In terms of initially creating your tempo page as a functional page, all you have to do is publish your application and make your data store record that you want to use for it visible to the tempo in the tempo sub-menu.

o   From there, it can pull a variety of different options, such as news, tasks, records, reports and actions as mentioned above. Based on what you want to show in the tempo, it would correlate to a certain tab. If you wanted to show a dynamic report you would just have to set it visible to the tempo and it’ll appear in the reports section for example.

 ·   Alternate Rules:

o   Not exactly sure how this would apply to tempo, since there aren’t really any alternative ways to my knowledge to making a tempo but if anyone has any suggestions feel free to add them.

·   Advantage/drawbacks:

o   The main advantage of tempo is that the main functionality that it gives you can really bring together several different business applications and processes that you’ve been developing and gives the end user several different intuitive options.

o   The main drawback is that the 5 menu options that tempo gives you are all that you can work with. You can never add anything more than news, tasks, records, reports, and actions.

·   Associated Topics:

o   The main associated topics would be the different parts of the tempo, as discussed earlier, as well as functionality you can make within the tempo that is accessed from your designer console, such as records or process models since they constitute your actions.

**Designing Drillable Reports**

**Definition:**

* A drillable or drilldown report is a report that allows users to explore multidimensional data by clicking on a data element which then reveals a finer level of detail of the data.

**Purpose:**

* Allows users to view different levels of data easily.

**Implementation Example (what)**:

* A pie chart that displays employees by status, where clicking a status category in the pie chart displays a grid of all employees with that status.

**Implementation Example (how):**

* Link to related Interface Pattern: [Configure a Chart Drilldown to a Grid](https://docs.appian.com/suite/help/20.4/recipe-configure-a-chart-drilldown-to-a-grid.html)

**Alternate Rules:**

* [Drilldown Report Pattern](https://docs.appian.com/suite/help/20.4/drilldown-report-pattern.html)
* [In-Place Drilldown Pattern](https://docs.appian.com/suite/help/20.4/drilldown-pattern.html)
* [Grid With Detail Pattern](https://docs.appian.com/suite/help/20.4/grid-with-detail-pattern.html)

**Advantage/drawbacks:**

* It eliminates the need for multiple visualizations of the data by allowing a user to explore different layers of the data within one visualization.

**Associated Topics:**

* Read-Only Grid
* Pie Chart
* Bar Chart
* Column Chart

**Linking Records and Reports**

**Definition:**

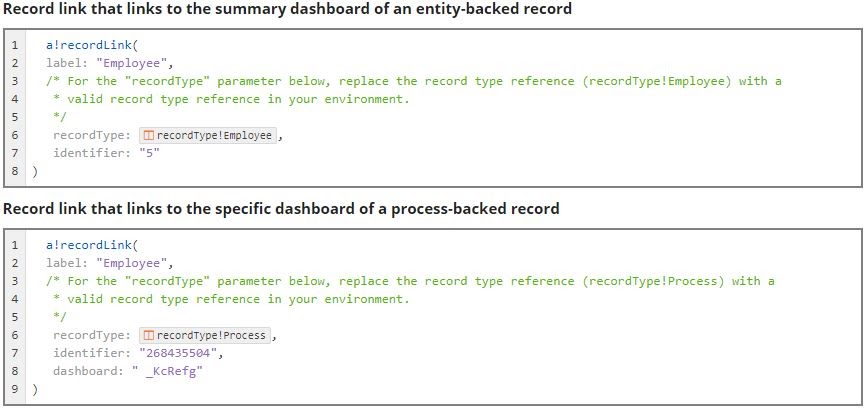
* ***Record Links*** define a link to a record view.
  + <https://docs.appian.com/suite/help/20.4/Record_Link_Component.html>
* ***Report Links*** define a link to a report view.
  + <https://docs.appian.com/suite/help/20.4/Report_Link_Component.html>

**Purpose:**

* Provide clickable links of records or reviews to users.

**Implementation Example (what):**

**Implementation Example (how):**





**Alternate Rules:**

* Record links that reference “process-backed” records use the dashboard parameter which is not required with an “entity-backed” record.

**Advantage/drawbacks:**

* Record links are usable in tags, charts, grids, hierarchy browsers, images, link fields, milestones, pickers, and rich text.
* Report links are usable in charts, grids, hierarchy browsers, images, link fields, milestones, pickers and rich text.

**Associated Topics:**

* Link fields
* Report type
* Record type

**Sub-process**

Definition:

* A process model that is configured in a parent process model as a node
* can be run either synchronously (finishing before the parent process does) or asynchronously (completing at any time - parent PM progress without waiting the sub process).

Purpose:

* for any functionality common to your process models.

Implementation Example (what):

Implementation Example (how):

Alternate Rules:

Associated Topics:

**Import/Export**

Definition:

Purpose:

Implementation Example (what):

Implementation Example (how):

Alternate Rules:

Associated Topics:

**Roles in Appian Team**

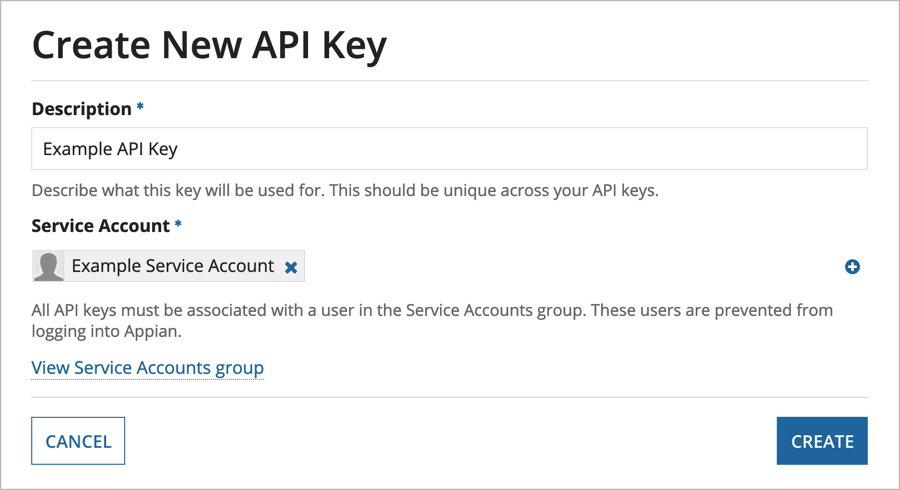
* Analyst - services between business users and developers and ensure requirements are clear for everyone.
* Designer - build, configure Appian application, create RDB, deploy the application for test or development
* Lead Designer - design the solution that meets the requirements from the analyst (know how to make maintainable, scalable and intuitive applications)
* Developer
* Tester
* Scrum Master
* Product Owner

**Application Patches**

* Definition:
  + A type of package that contains new or updated objects, which a developer deploys when introducing an update to an existing application in the target environment.
    - A **package** is a collection of Appian application changes that a developer can deploy to another environment.
* Purpose:
  + Used to introduce a new update to an existing application
  + Helpful for deploying bug fixes or enhancements
* Implementation Example (what):
  + Deploying a patch that adds a new “Service Request” record type and an “Add New Request” action page to to a site within an Appian application
  + Deploying a patch that fixes a bug within a process model where updating an employee’s details instead creates a new employee record in the database.
* Implementation Example (how):
  + Like all packages, an application patch can be deployed 3 different ways:
    - **Compare and Deploy** - Directly deploy applications, patches, plug-ins, and database scripts with guided steps.
    - **Export and Import** - Manually import and export applications and patches between environments
    - **Automated Deployment Manager** - Customize your own deployment process with this tool.
  + Further implementation details on how to deploy patches and other packages can be found here: [Deploy to Target Environments](https://docs.appian.com/suite/help/20.4/Deploy_to_Target_Environments.html)
* Alternate Rules: N/A
* Advantages:
  + Can deploy bulk patches to multiple applications
* Associated Topics:
  + [Deploying Applications](https://docs.appian.com/suite/help/20.4/deploying-applications.html)
  + [Types of Packages](https://docs.appian.com/suite/help/20.4/prepare-deployment-packages.html#types-of-packages)
  + [Deploy to Target Environments](https://docs.appian.com/suite/help/20.4/Deploy_to_Target_Environments.html)

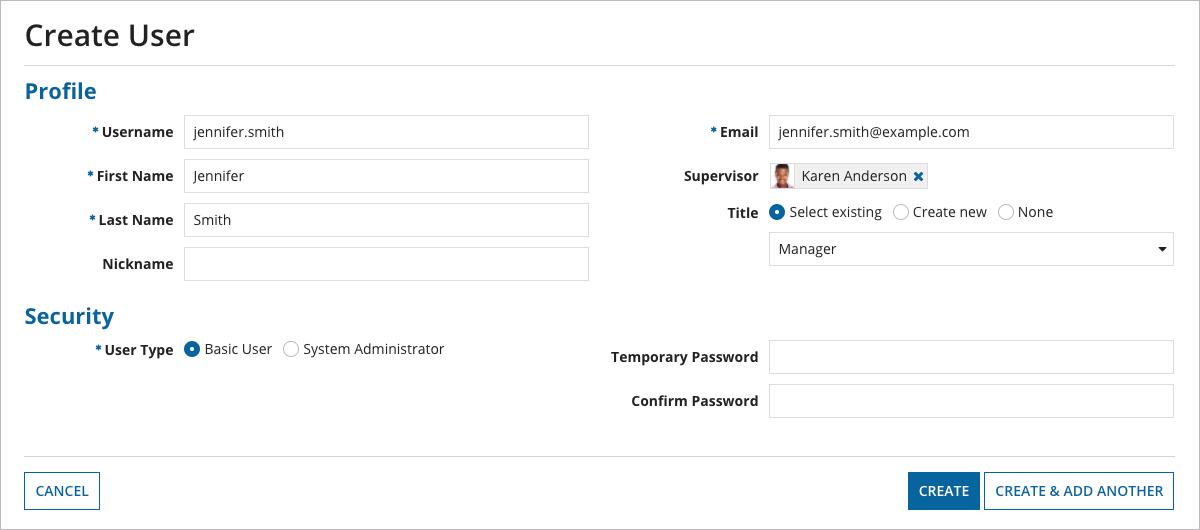
**Admin Console**

* Definition:
  + The web interface where System Administrators can update certain configuration properties within the Appian environment.
  + Configuration properties are divided into the following categories: System, Authentication, DevOps, Integration, Monitoring, and the Import and Export of Admin Console Settings.
  + Pages related to **System** administration:
    - **Branding** - lets you manage the name, logos, and colors that appear throughout the Tempo interface.
    - **Data Retention** - lets you manage news entry deletion and the periodic cleanup of deployment packages.
    - **File Upload** - lets you manage real-time virus scanning and control what types of files are allowed to be uploaded based on their extension.
    - **Internationalization** - allows you to set the primary language and time zone shown to users, and also the calendar types and languages enabled for the site.
    - **Mobile** - allows you to manage settings for your organization’s mobile devices.
    - **Permissions** - allows you to control users’ actions within various features on Tempo.
    - **Plug-ins** - lists all plug-ins that are available in the environment.
    - **Sign-in page links** -  allows you to add custom links to the sign-in page.
    - **Sites Typeface** - allows you to add and preview up to 9 custom typefaces for your sites on both web and mobile.
    - **User Start Pages** - allows you to configure which pages users start on when they first log into Appian.
  + Pages related to **Authentication** administration:
    - **API Keys** - allows you to create and manage Service Accounts and API keys, which can be used to invoke Appian Web APIs.
    - **Appian Authentication** - allows you to control password strength requirements and password expiration policies.
    - **LDAP Authentication** - allows you to configure Appian to authenticate users against an external directory server, like Microsoft Active Directory, via LDAP rather than its native authentication.
    - **Maintenance Window** - allows you to set a period of time to deploy application updates to your environment and write a message to display in a banner at the top of your site.
    - **SAML Authentication** - allows you to configure Appian to authenticate users against external SAML identity providers, like Microsoft ADFS or Shibboleth, rather than against Appian's native authentication.
    - **Users** - allows you to create, view, and modify users in Appian.
  + Pages related to features that support **DevOps**:
    - **Deployment** - allows you to manage whether to allow test values to be imported with design objects and whether to allow database schema changes through data stores.
    - **Health Check** - allows you to set up, schedule, and run Health Checks on your Appian environment.
    - **Infrastructure** - displays all the Appian environments in your organization and lets you control how those environments interact with each other.
  + Pages related to **Integration** administration:
    - **Certificates** - allows you to manage two types of SSL certificates in the Admin Console: Client Certificates and Trusted Server Certificates.
    - **Data Sources** -  lets you integrate Appian with external databases using a JDBC connection by adding, updating, and removing named connection configurations called data sources.
    - **Document Extraction** - allows you to enter your Google Service Account key information in order to use Document Extraction features in Appian.
    - **Email** - allows you to enable or disable the ability for Appian to send email.
    - **Embedded Interfaces** - allows you to manage origins and themes for embedding interfaces on external web sites.
    - **HTTP Proxy** - allows you to configure a proxy server for outgoing HTTP and HTTPS connections.
    - **Legacy Web Services** - allows you to manage processes exposed as web services (deprecated).
    - **Microsoft Office** - allows you to configure Appian's Task Viewer Add-in for Microsoft Outlook.
    - **Third-Party Credentials** - allows you to manage credentials to external systems.
  + Pages related to **Monitoring** your Appian installation:
    - **Current User Activity** - allows you to see which users are currently active on your site.
    - **Document Reports** - shows usage information about documents in the system such as general statistics, user statistics, and the number of uploads and downloads that occurred in the last 10 days.
    - **Import History** - allows you to see all the imports that have occurred on the system during the last 30 days.
    - **Rule Performance** - allows you to see the historical performance of all of the rules in the system.
  + Buttons related to the **Import and Export of Admin Console Settings**
    - **Export** - opens the Export Settings dialog that shows a list of all available admin console settings available to be exported.
    - **Import** - opens the Import Settings dialog that allows you to import an application package and an administration console package together.
* Purpose:
  + Allows system administrators to configure system, authentication, and integration settings, as well as monitor system activity.
* Implementation Example (what):
  + One way system administrators use the Admin Console is to create an API Key for invoking Appian Web APIs.
* Implementation Example - Creating an API Key:
  + Open the Admin Console.
  + Navigate to the API Keys page in the Authentication section of the Admin Console.
  + Click the **Create** button. This will prompt you to provide a unique description and select a service account to associate with the key.

* If you need to create a new service account, you can easily do so by clicking the plus icon to the right of the Service Account picker on the API key creation modal. This prompts you to provide a username.
* When the key is generated, the designer should immediately copy the value and store it externally. Appian will never show the value of the API key a second time.
* Alternate Rules: N/A
* Advantages/drawbacks:
  + Simplifies the configuration of Appian environment settings by separating them into related categories for easy access.
* Associated Topics:
  + [System Administrators](https://docs.appian.com/suite/help/20.4/User_Management.html#manage-user-rights-and-security)
  + [API Keys](https://docs.appian.com/suite/help/20.4/Appian_Administration_Console.html#api-keys)

**Users**

* Definition:
  + Any individual who can access and use any of the Appian interfaces (such as Tempo, Designer Interface, or the System Administrator Console)
  + Similar to user accounts in other web applications, an individual must log in as a user in order to use any of Appian’s features
  + The access rights granted to a user are determined by the following factors:
    - **User Type** - Assigning a user to one gives the user a set of rights to perform specific actions in the system.
      * **System Administrators** - Have administrative rights to all components, and can access the System Administrator Console.
      * **Basic Users** -  Start out with Viewer rights to top-level objects and interface tabs, but can have additional rights granted to them for specific objects or modified by System Administrators.
    - **User** **Role** - Adding a user to one gives the user access to specific interfaces in the Appian system
      * **Application User Role** - gives a user access to Tempo, sites, and embedded interfaces
      * **Designer Role** - gives a user access to the Designer Interface, Tempo, and any applications exposed through an end user environment
    - **Groups** - Used to organize users
      * Can contain both user members and group members
      * Give users access to specific areas of Appian using user roles.
      * Two basic groups for any app (prefixed with App name)
        + All Users - parent group for all (contains all users and groups)
        + Administrators - use this group to secure the All Users group
* Purpose:
  + The concept of users in Appian allows for individuals to be granted (or denied) certain access privileges
  + Through the use of User Types, User Roles, and Groups, Appian Developers can effectively control a user’s experience within Appian
* Implementation Example (what):
  + In our EMS project, we created users belonging to the HR Users group who were in charge of adding employees to the database. We also created users belonging to the HR Managers group who were responsible for approving/rejecting new employees.
* Implementation Example - Creating a New User:

1. Navigate to the **Users** page of the Appian Administration Console. The Users grid is displayed.
2. Click the **Create** button in the toolbar above the grid. The **Create User** dialog is displayed.  
     
   
3. Enter the user's **Username**. It must not match an existing username regardless of case. For example, if "john.doe" already exists, you cannot enter "John.Doe". Appian usernames are case sensitive when the user logs in. A username can only contain up to 255 ASCII letters (a-z, A-Z), numbers, and the special characters listed below.
   * At symbols (@)
   * Periods (.)
   * Underscores (\_)
   * Hyphens (-)
4. **Note**: If you plan on creating an application that restricts user visibility, be sure to create a username that is not personally identifiable, as usernames are still visible within the system even when a user's Contact Information and Display Name are restricted. See also: [User Profile Visibility](https://docs.appian.com/suite/help/20.4/User_Profile_Visibility.html)
5. Enter the remaining basic information for the user:
   * First Name
   * Last Name
   * Nickname (Optional)
   * E-mail Address (Must include the **@** symbol and a domain such as **.com**)
   * Supervisor (Optional)
   * Title (Optional)
6. Select one of the following options from the **User Type** selection:
   * **Basic User**: All users that are not system administrators. See below: [Account Rights for Basic Users](https://docs.appian.com/suite/help/20.4/User_Management.html#account-rights-for-basic-users)
   * **System Administrator**: Users that have access privileges to all tools and capabilities in Appian, can edit user roles, and create new administrators. System administrators are automatically added to the Designers group through a membership rule, but do not inherently have the [designer role](https://docs.appian.com/suite/help/20.4/User_Roles.html#designer-role).
7. (Optional) Enter a **Temporary Password** and re-enter it in the **Confirm Password** field. If a password is not entered, it is automatically generated and sent to the user in an email. Regardless of whether or not you specify a password here, the password will be marked as temporary and the user will need to change it the next time they sign in. **Note**: Passwords may be constrained to certain complexity or length requirements by your Appian administrator. See also: [Appian Authentication](https://docs.appian.com/suite/help/20.4/Appian_Administration_Console.html#appian-authentication).
8. Click **Create** to create the user and return to the Users grid, or click **Create & Add Another** to create the user and continue creating new users in the Create User dialog.

* Related Functions:
  + touser(*value*) - Converts a value (such as a username) to a User.
  + user(*username, property*) - Returns a selected property from a user’s profile and preferences.
  + loggedInUser() - Returns the current user logged into the application.
  + isusermemberofgroup(*username, groupId*) - Identifies whether or not a user belongs to a group.
* Advantages/drawbacks: N/A
* Associated Topics:
  + [User Management](https://docs.appian.com/suite/help/20.4/User_Management.html)
  + [User Roles](https://docs.appian.com/suite/help/20.4/User_Roles.html)
  + [Group Management](https://docs.appian.com/suite/help/20.4/Group_Management.html)

**Local Variables using load() and with()**

Definition:

* load()
  + Lets you define local variables within an expression for an interface and evaluate the expression with the new variables, then re-evaluate the function with the local variables' values from the previous evaluation.
  + Variable value is re-evaluated on page refresh.
* with()
  + Lets you define local variables within a function and evaluate the expression with the new variables.
  + Variable value is re-evaluated whenever the user interacts with the interface.

Purpose:

* load() is used in expressions for interfaces to allow for user interaction on the interface. Every time that the page or interface is initially “loaded”, the variable will be reevaluated.
* with() differs from load(), because it’s declared variables will be evaluated on each initial interaction of the variable. This could be used when dynamically presenting data in an interface and you need to evaluate some expression or variable on a user’s click or typing.

Implementation Example (what):

* In an interface example, you can use these functions to allow dynamic re-evaluation when presenting data in a grid, and wanting to filter data in a text box without a search button. You could use the with() to have your grid update upon any typing of the filter box
* .Implementation Example (how):
* These functions can be called using the same format as a!localVariables(), with the variable name and its corresponding expression being written inside the parentheses of the “load” or “with” functions.
* Ex: with( local!withVariable : *someExpression* )

Alternate Rules:

* a!localVariables() handles the actions of both of these functions combined, plus extra options in terms of controlling refresh options. It removes the additional calls of two functions, while providing the same functionality.

Advantage/drawbacks:

* Advantage: You can be more explicit to other users about what your variable is doing with the interface
* Disadvantage: There is already a function that encompasses both of these functions, so it would be more relevant to use a!localVariables() instead

Associated Topics:

* a!localVariables()
* interfaces

**Process Management & Debugging**

Definition

* Observing processes in systems and applications, to better identify errors in process models and tasks

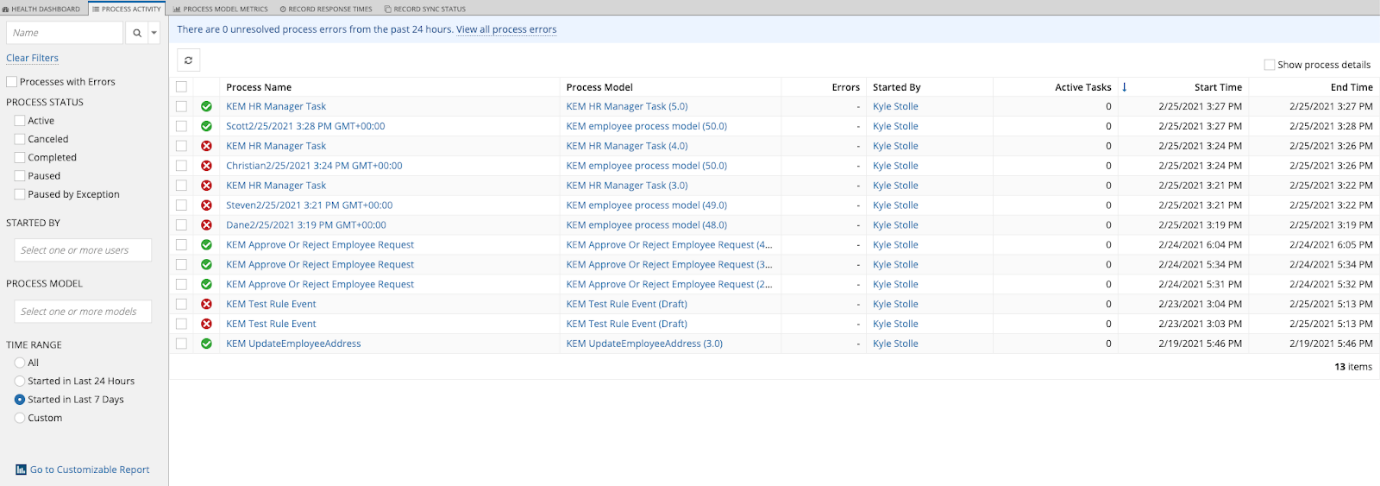
Purpose

* Allows you to monitor application processes and find any potential errors within these processes or tasks

Implementation Example

* Once you’ve selected your application, you can click on the monitoring tab next to objects to pull up the health dashboard
* From there you can select the “process activity” tab which will bring up a detailed grid of process information for your application





* In this view you can filter your processes or you can select individual ones to do further debugging in the process model view

Advantages/Drawbacks

* Advantages
  + Allows users to view all processes related to specific applications
  + Allows users to filter for certain processes.
    - For example, identifying all currently active processes
  + Allows users to select and further inspect individual processes

Associated Topics

* Process Models

**Appian Web Service Integration**

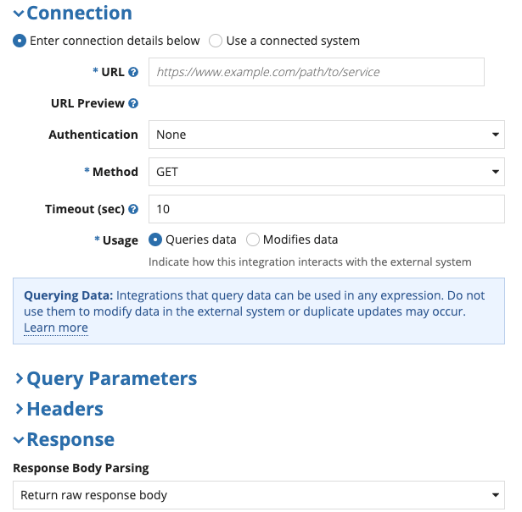
Definition

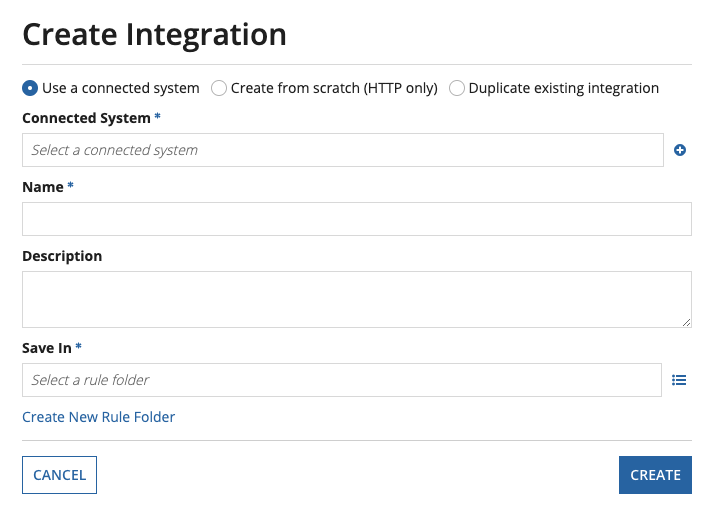
* An Integration Object allows Appian to access data from a 3rd party system.
* When configuring an HTTP Integration, Appian sends an HTTP request to the 3rd party system, which will then process the request and send back a response.
* To call a 3rd party Web API, you must specify the base URL

Purpose

* Allows Appian to connect to an external system and access its web services.
* Gives expression rules, interfaces, or process models in Appian the ability to call Web APIs from another system.

Implementation Example

* In your objects view on the cloud suite, click on new and select integration. The following screen should appear



* If you’re using a Connected System, specify which one and then fill out the “Name” and “Save In” parameters
* If you’re creating one from scratch just fill out the “name” and “save in” parameters
* Once you create it, you can then configure the various aspects of the integration object. In the above HTTP integration object instance, you can configure the connection, query parameters, headers, and response format
* Make sure to select the correct usage (Queries data or Modifies data) based on how the integration will be used

Advantages/Drawbacks

* Advantages:
  + Allows Appian to connect to an external system and access its web services.
  + If using an HTTP Integration, you can specify the HTTP request’s query parameters, header, and body using rule inputs.
  + Easy to test using expression rules or interfaces
* Drawbacks:
  + If you do not use a Connected System when creating multiple Integrations for a single system, any changes to the Integrations’ base URL or authentication details must be done manually for each and every Integration Object.

Associated Topics

* Web APIs
* Connected Systems

**Web API**

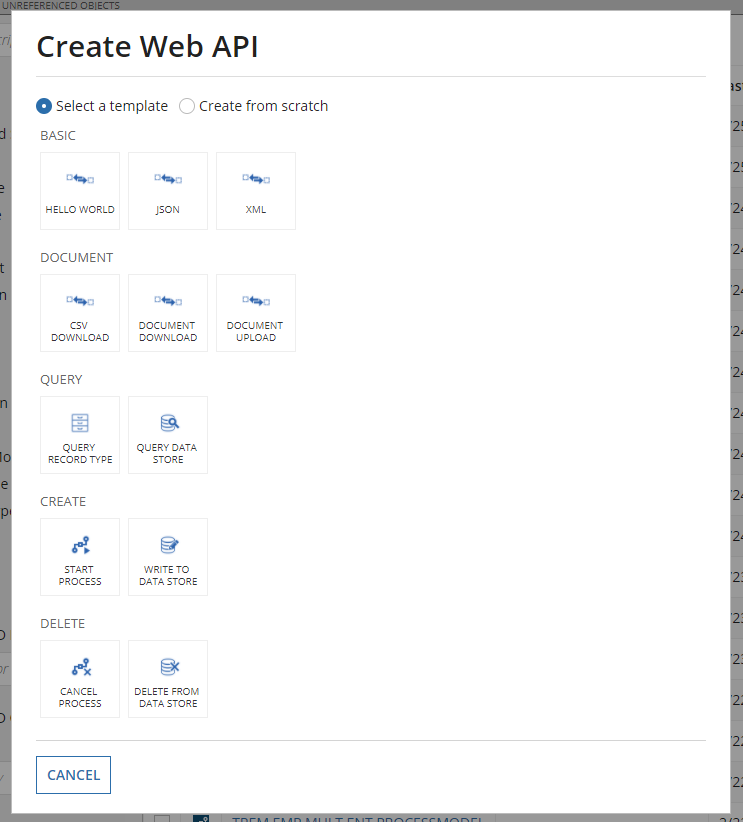
Definition:

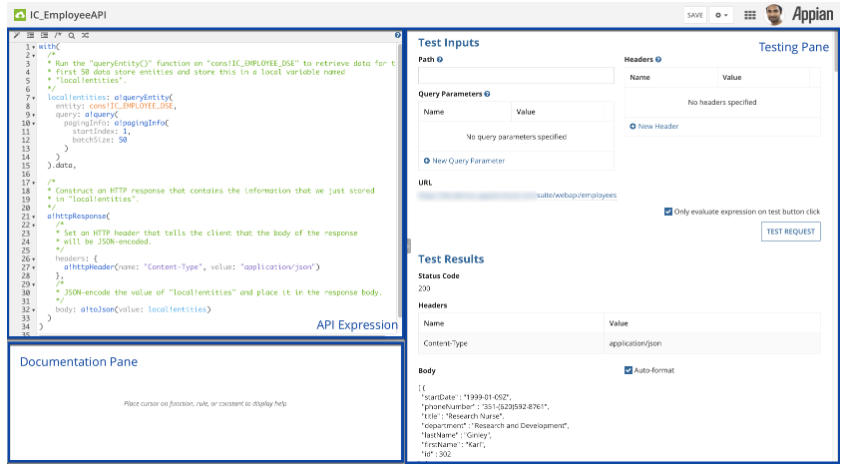
* A Web API is an object that exposes Appian data and services to outside systems.
* They include an endpoint that can be called by other systems.
* Outside systems call an Appian Web API using an HTTP request, and the Web API in turn sends back an HTTP response.
* When called, Web APIs are passed values via query parameters, headers, a body, or any combination of the three.

Purpose:

* Allows an outside system to access functionality within an Appian application.
* Lets an outside system perform CRUD operations in Appian, such as:
  + Querying a record type
  + Starting a process
  + Deleting from a data store
  + etc.

Implementation Example:

* Upon creating a new Web API object, you will see the following options:

* These are the templates that are given to you beforehand, that follow the main CRUD methods, as well as giving options to send basic data as XML or JSON.
* Once you choose a template, or if you decide to build from scratch you will be shown the same screen where you can name the object and configure the endpoint that will be used to access this specific API.
* Once that configuration is complete, you will see this window above where you will be able to define the query you are trying to make, send test inputs as query parameters, and see the data being passed in the header, body, and status code.

Alternate Rules: N/A

Advantage/drawbacks:

* The advantage to this is the flexibility that the Web API brings when it comes to sending and passing data. As long as you have the endpoint, you will be able to access this data without actually being a part of the Appian application. This allows for a broader access of systems that can connect to Appian and take advantage of its stored data.

Associated Topics:

* Connected Systems
* Appian Web Service Integration

**Connected Systems**

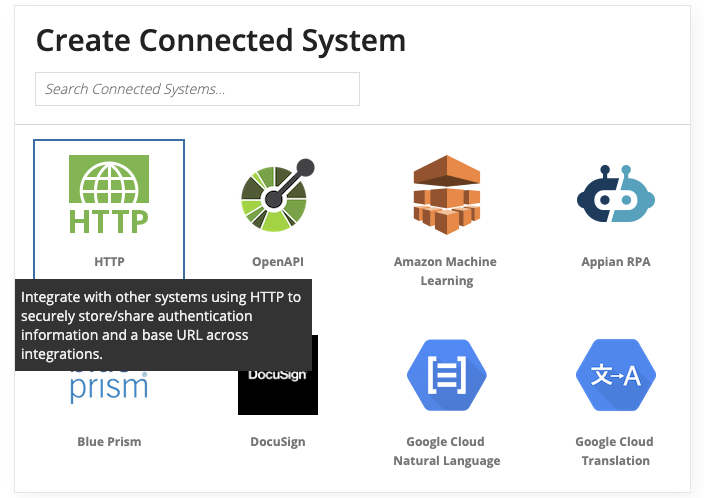
Definition:

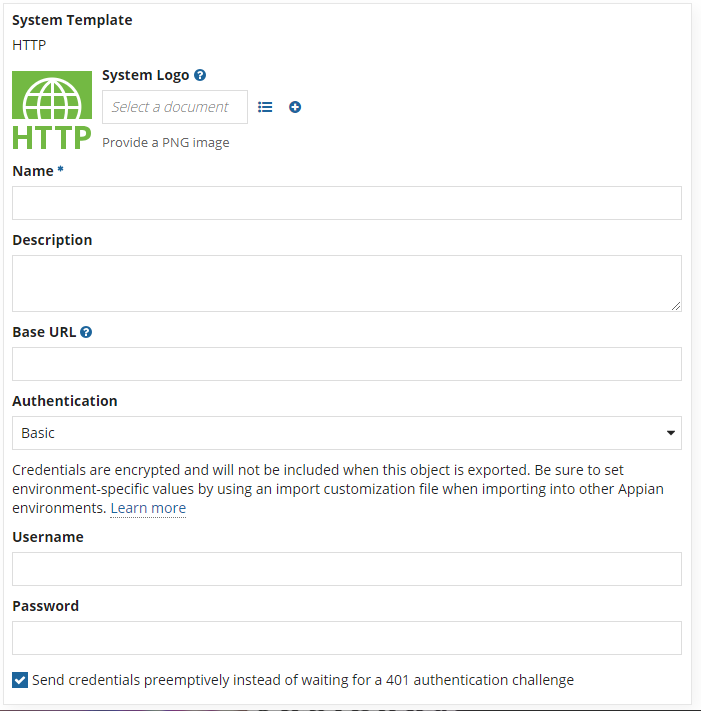
* A Connected System object represents an external system that is integrated with Appian.

Purpose:

* Allows you to share a base URL and authentication details across multiple integrations.
* Allows you to group together Integration Objects that belong to the same external system.

Implementation Example:

* Upon creating a new Connected System object, you will be presented the following connected system templates
* These different templates are what your connected system will be based off of, and they each will provide your integrations that use this system with the fields specific to that third party system.
* We have mostly been using the HTTP system to connect our integrations.



* After choosing a template you will be shown this screen above, where you can name and define the base URL that will be used across the Connected System.
* You can then configure the type of authentication you want to persist throughout the object.
* We used an API Key authentication, where we were given a random key that we used to create a Service Account user that could be added to our User Group, thus granting that entire group access to the key and the Connected System as a whole.
* This Connected System can then be used to connect all integration objects throughout your application.

Alternate Rules: N/A

Advantage/drawbacks:

* Allows Integration Objects that belong to the same external system to share a base URL and authentication details.
* Allows you to reconfigure connection details for several Integration Objects all in one place.
* Lets you quickly generate new Integration Objects by using a preexisting Connected System.
* Gives Appian designers access to additional authentication options.

Associated Topics:

* Web Integration
* Appian Web Service Integration