Chapter 1  Introducing Infogix Data3Sixty

What is Infogix Data3Sixty? ............................................................... 17
Infogix Data3Sixty System Overview ................................................. 17
   Environments ............................................................................... 17
   Pipelines ...................................................................................... 18
      Creating Pipelines ............................................................... 18
      Using Pipelines ...................................................................... 18
Data Stages ....................................................................................... 19
   Data Store.............................................................................. 19
   Analysis ...................................................................................... 19
   Analytic Model ....................................................................... 19
   Data View............................................................................... 19
   Process Model ....................................................................... 19
   Dashboard ............................................................................. 20
   Case Store............................................................................. 20
   Rule Library ........................................................................... 20
Working across Environments ........................................................... 21
How will you use Infogix Data3Sixty? ................................................ 23
Other Sources of Information ............................................................ 24
Contacting Customer Support ........................................................... 24

Chapter 2  Part 1: Acquisition, Preparation, and Operationalization

Chapter 3  Account Activation

Logging in with Enterprise SSO ................................................... 28
Account Passwords ..................................................................... 28

Chapter 4  Data Store Configuration

Data Store Type ............................................................... 29
Internal Data Stores ................................................................... 29
   Store Repository Types .................................................................. 29
External Data Stores .................................................................... 30
   Store Repository Types ............................................................... 30
   Infogix Data3Sixty Cloud Version ........................................... 30
   Infogix Data3Sixty Enterprise Edition ...................................... 31
Supported File Layouts ................................................................. 31
  Delimited Layout ................................................................. 31
  Line Layout ........................................................................ 33
  Additional File Layouts .......................................................... 34
  Microsoft Excel Layout .......................................................... 34
  XML Layout ........................................................................ 35

Creating Data Stores ......................................................................... 38
  Creating External Data Stores ...................................................... 38
    Direct Upload of File .............................................................. 38
    S3, HDFS, or Alternative File System Upload ............................. 38
    Database ........................................................................ 40
  Creating Internal Data Stores - External Applications.............. 43
  Creating Internal Data Stores - Data Stage Outputs ................. 44
    Creating Internal Database Data Stores - for use with Case Stores .............................................. 44
    Creating Data Stores that allow Ad Hoc Queries...................... 45

Data Store Fields ............................................................................. 46
  Field Types ........................................................................ 46
  Semantic Type ....................................................................... 47
  Default Value ....................................................................... 47
  Settings for Numeric Fields .................................................... 47
  Settings for Date, DateTime, and Time Fields ......................... 50
  Settings for String Fields ....................................................... 51
  Generating Fields .................................................................. 51
  Importing Fields ................................................................... 51
  Testing Fields ....................................................................... 52
  Setting up a Single Field Data Store for Parsing ..................... 52
  Encrypted Fields ................................................................... 52
  Secure Fields ....................................................................... 53
  Code Sets ........................................................................... 54
  Synonyms ........................................................................... 55

Transform .......................................................................................... 56
  Script Inputs ....................................................................... 56
  Script Outputs .................................................................... 57

  Using the Script Input and Script Output tabs to handle Javascript’s Numeric Limitations .................. 57
  Script Rules tab ................................................................... 58

Identity ........................................................................................... 58

Indexes (Internal Database Data Store only) ................................. 58
Screens (Internal Database Data Store only) ........................................ 59
Search Screens ............................................................................. 59
View/Edit Screens ......................................................................... 60
Query Filter (Internal Database Data Store only) ............................ 60
Using a Filter Expression and a Super Group ............................... 61
Using a Filter Expression with User Profile Fields ....................... 61
Using Multiple Query Filters ....................................................... 62
Edit Restriction (Internal Database Data Store only) ..................... 62
Edit Restriction Example .............................................................. 62
Links ............................................................................................. 63
Profile tab ...................................................................................... 63
    Profiling Parameters .............................................................. 64
    Executing a Data Store for Profiling ....................................... 67
Other tab ....................................................................................... 68
    Retention Example 1 .............................................................. 68
    Retention Example 2 .............................................................. 68
    Retention Period Over Time .................................................. 69
    Retention Period Data Flush .................................................. 70
Handling Data Store Contents ..................................................... 71
    Using the Upload File feature ............................................... 71
    Executing Dependent Data Stages After Upload ....................... 71
Using the Delete All Data Feature ............................................... 71
    Affect on other Data Stages ................................................... 71
Viewing Data Store Contents ....................................................... 72

Chapter 5  Rule Library

Details tab .................................................................................... 73
Rules tab ....................................................................................... 73
    Data Quality Rule Group ....................................................... 73
    Completeness Rule ............................................................... 74
    Consistency Rule ................................................................. 74
    Value Conformity Rule ........................................................ 75
    Type Conformity Rule ........................................................... 76
    Checks ................................................................................... 76
    Regular Expression Rule ....................................................... 76
    Timeliness Rule ..................................................................... 76
    General Expression Rule ....................................................... 77
    Check Expression ................................................................. 77
Chapter 6  Using the Analysis Designer

Creating an Analysis ............................................................. 80
  Prerequisites ...................................................................... 81
Saving an Analysis ............................................................... 81
Saving an Analysis Definition .............................................. 81
Analysis Designer Nodes ...................................................... 81
  Input/Output Nodes .......................................................... 81
    Data Store Input ............................................................ 81
    Parameterized Data Loads ............................................. 84
    Data Store Output .......................................................... 90
    Case Store Input ........................................................... 93
    Case Store Output ......................................................... 93
    Custom Completion Status .......................................... 94
    Sqoop In ........................................................................ 97
    Sqoop Out ..................................................................... 98
    Output Plugin .................................................................. 100
    Kinesis ......................................................................... 100
    Excel Output .................................................................... 100
Including other Analyses within an Analysis .......................... 104
  Placeholder Input ............................................................ 104
  Placeholder Output ......................................................... 105
  Nested Input Analysis ..................................................... 105
  Nested Output Analysis .................................................. 107
Enhance Nodes .................................................................... 108
  Sample ............................................................................ 108
  Select ............................................................................. 108
  Sort ................................................................................. 109
  Rename ........................................................................... 109
  Filter ................................................................................. 109
  In .................................................................................. 109
  Not In .............................................................................. 111
  Plugin .............................................................................. 113
  Compute Node ................................................................. 113
  Nested Analysis .............................................................. 113
Combine Nodes .................................................................................... 114
Join ........................................................................................................ 114
Union .................................................................................................... 115
Co-Group ............................................................................................ 116
SQL......................................................................................................... 117
Execute Query in DB ........................................................................... 118
Group .................................................................................................... 122
Shape Nodes ....................................................................................... 125
Group By ............................................................................................. 125
Distinct .............................................................................................. 126
Flatten .................................................................................................. 126
Bin.......................................................................................................... 127
Script..................................................................................................... 128
Auto Number ....................................................................................... 132
Reshape ............................................................................................... 132
Split Node ............................................................................................ 136
REST API Call Node ........................................................................... 138
Check Nodes ........................................................................................ 141
Profile Data Node ................................................................................ 141
Completeness Check Node ............................................................... 144
Consistency Check Node ................................................................. 145
Recon Check Node ............................................................................. 146
Timeliness Check Node ...................................................................... 151
Value Conformity Node ..................................................................... 152
Type Conformity Node ....................................................................... 153
Execute Rule Library .......................................................................... 154
Profile Data Legacy Node ................................................................. 155
Analytics Nodes .................................................................................. 158
Analytic Models .................................................................................. 158
How to use an Analytic Model ......................................................... 158
Measures used for Evaluation ......................................................... 162
Profit ...................................................................................................... 163
Segmentation ....................................................................................... 167
Classification ...................................................................................... 170
Regression ........................................................................................... 175
Recommendation ............................................................................... 179
Forecast ............................................................................................... 184
Text Analytics nodes .......................................................................... 187
Identify Key Phrases .......................................................................... 187
Chapter 7  Creating Data Views

Linking Multiple Data Stores ........................................................................ 204
Data View Properties ................................................................................ 204
Details ........................................................................................................ 204
   Storage Type ........................................................................................ 204
   Table Name Prefix ............................................................................... 205
Dimensions ................................................................................................. 205
   Fields ................................................................................................... 206
Data View Field Settings ............................................................................ 206
Encrypted Fields ......................................................................................... 210
Secure Fields .............................................................................................. 211
Computed Fields ......................................................................................... 212
Drill fields .................................................................................................. 212
Group Fields ............................................................................................... 213
Geo Fields .................................................................................................. 213
Chapter 8  Executing Data Stages

What it Means for different Data Stages to “Execute” .............. 226
  Data Store ........................................................................ 227
  Data View ......................................................................... 227
  Analysis ........................................................................... 229
  Process Model ................................................................... 230
  Process Model ................................................................... 230
  Performing or Scheduling Executions ............................... 232
    Manual Execution .......................................................... 233
    Scheduled Execution ..................................................... 233
    Process Models ............................................................ 233
  Viewing a Data Stage’s Executions History ....................... 233
  Searching for Executions ................................................. 234
Chapter 9 Creating Process Models

Creating a Process Model ............................................................... 236
Prerequisites ........................................................................ 237
Process Model Variables ........................................................... 237
Passing Variables from Node to Node ........................................ 237
System defined variables ..................................................... 237
User defined variables ......................................................... 237
Execution Result Codes ...................................................... 240
Execution Result Types ....................................................... 241

Process Model Nodes ..................................................................... 241
Event Nodes .............................................................................. 241
Stage Start Event................................................................. 241
External Start Event ............................................................. 242
Stage Intermediate Event .................................................... 242
Timer Intermediate Event ..................................................... 242
Success Intermediate Event ................................................ 242
Error Intermediate Event...................................................... 242
Cancel End Event ................................................................ 242
Error End Event ................................................................... 243
Success End Event.............................................................. 243

Tasks.......................................................................................... 243
Execute Stage Task ............................................................. 243
Execute Process Task.......................................................... 243
Notify from Data Store Task ................................................. 244
Notify Task ........................................................................... 244
Script Task ........................................................................... 246
Delete From Data Store Task ............................................... 247

Gateways................................................................................... 248
Exclusive Gateway .............................................................. 248
Inclusive Gateway............................................................... 249
Parallel Gateway................................................................. 250

Basic Process Modeling Example ............................................. 250
Process Model Termination on Failure ..................................... 252
Using Multiple Processes within a Process Model ............... 253
Chapter 10 Part 2: Data Exploration

Chapter 11 Building a Dashboard

Where should I build my Dashboard? ............................................. 258
Where should I save my Dashboard? ............................................. 258
Pipeline Structure, Data Views, and Permissions ...................... 258
Building a Dashboard for Case Management .......................... 259
Dashboard Elements .................................................................... 260
Charts and Tables ....................................................................... 261
Fields ....................................................................................... 261
Computations .......................................................................... 261
Card Dashlets ........................................................................... 263
Adding, Editing, and Deleting Cards ........................................ 263
Configuring how Cards are Displayed ................................. 264
Text Elements ........................................................................... 264
Place Holders ........................................................................... 265
Filters ....................................................................................... 265
Filter Values By Parameter .................................................... 265
Selection Type ......................................................................... 266
Cascading Filters ...................................................................... 267
Targeting a Specific Dashlet ..................................................... 268
Filter selections ....................................................................... 268
Web Content ............................................................................. 269
Spacers ................................................................................. 269
Dashboard Sheets ...................................................................... 269
Advanced Dashboard Features ...................................................... 269
Changing Chart Properties ........................................................ 269
Changing Chart Display .............................................................. 269
Drilling to another Data Stage .................................................. 270
Chart Filters ............................................................................. 270
Multi-Series Charts for Color Coding ........................................ 270
Applying Thresholds ................................................................. 270
Features Common to all Thresholds ........................................... 271
Highlight All Points feature ....................................................... 271
Process All Rules Feature ......................................................... 271
Thresholds that can be applied to most Charting Types .............. 272
  Single Point Rule .................................................................. 272
  Multipoint Rule ..................................................................... 272
  Multipoint Rule parameters .................................................. 273
  Expression Rule ................................................................... 275
Control Chart Thresholds ........................................................... 276
  Western Electric Rules .......................................................... 276
  Nelson Rules ....................................................................... 280
  Tool Tips and Overlapping Threshold Rules Violation .......... 284
  Standard Deviation Date Filter Override for Control Charts. 284
  Applying Query Criteria ....................................................... 285
  Setting up a Dashlet's Targets ............................................... 285
  Tooltip Customization ......................................................... 286
Other Dashboard Features ........................................................... 288
  Copying Dashlets ................................................................ 288
  Copying Sheets ..................................................................... 288
  Exporting a Dashboard .......................................................... 288
  Image export ....................................................................... 288
  Print View ........................................................................... 288
  Delimited File Export .......................................................... 289
  Undo/Redo when Viewing a Dashboard ............................... 289
  Refreshing a Dashboard ....................................................... 289
  Unmasking Secure Fields in Dashboards ......................... 289
Features Specific to certain Chart Types ......................................... 289
  Data Grids .......................................................................... 289
  Flex Width Columns ............................................................ 289
  Lock Columns ..................................................................... 290
  Show Tooltip ....................................................................... 290
  Pivot Tables ......................................................................... 290
  Custom Measures .................................................................. 290
  Dashboard Settings ............................................................... 291
  Sheet Settings ............................................................... 291
  Secure Access to Visualizer .................................................... 291
Styling a Dashboard ............................................................... 291
  Dashboard Fonts .................................................................. 291
  Setting a Global Font Family ............................................... 291
  Styling Font for Filter Dashlets ............................................ 291
  Styling Font for Filter Selection Dashlets ............................ 292
  Styling Font for Chart/Table Dashlets ................................. 292
Chapter 12  Using the Data3Sixty DQ+ Visualizer

Data3Sixty DQ+ Visualizer Features ..................................................... 294
Chart filters ......................................................................................... 294
Thresholds ......................................................................................... 294
Query criteria ...................................................................................... 295
Show Data Details .............................................................................. 295
Customizing Visualizer Properties ...................................................... 295
Color Tab ............................................................................................. 295
Chart Tab ........................................................................................... 296
Axes Tab ............................................................................................. 298
Grid Style Tab .................................................................................... 301
Style Tab ............................................................................................ 301
Data Label Tab .................................................................................... 301
Header Tab ........................................................................................ 302
Gauge Tab .......................................................................................... 302
Important Notes about the Infogix Data3Sixty Visualizer ................. 302
Visualizations are not saved ............................................................ 302
Saving a Visualizer State to a Dashboard ......................................... 302
Charts and Tables Only ...................................................................... 302
1 Sheet, 2 Chart Limit ....................................................................... 302

Chapter 13  Part 3: Sharing, Collaboration, and Controls

Chapter 14  Permissions and Collaboration Features

Sharing Data Stages ............................................................................. 304
Sharing in Pipelines ........................................................................... 304
Who to Share With: Super Groups ....................................................... 305
How to Grant Permissions ................................................................ 305
Pipeline Permission/Action Matrix ...................................................... 306
Permissions Applied at Multiple Levels .............................................. 308
Moving, Copying, or Duplicating Data Stages ..................................... 310
Moving a Data Stage ........................................................................... 310
Copying a Data Stage ......................................................................... 310
Duplicating a Data Stage .................................................................... 311
Assigning Watchers ........................................................................... 311
Who Can Watch: Users and Groups ................................................... 311
Collaboration Features ....................................................................... 312
Dashboard Annotations ............................................................. 313
Attaching Documents ................................................................. 313
Making Comments ..................................................................... 313
Comment Threads ..................................................................... 313
Pinning Favorite Dashboards to your Home Screen ....................... 314
Customizing the Explore Data Menu ............................................ 314

Chapter 15  Dashboard Controls

Notification Functionality ...................................................... 315
Control Parameters ............................................................... 315
When are Controls applied? ................................................ 316
Viewing Control Comments ................................................. 316
Basic Control Flow ............................................................... 317
Controls’ Interaction with Other Data Stage Configurations 318
Control Permissions ............................................................. 319
When to use a Control instead of just a Threshold .............. 319
Using a Custom Notification Process .................................. 319

Chapter 16  Part Four: Case Stores

Building Case Store Definitions ....................................................... 324
Case Store Tabs ........................................................................ 325
Details Tab ........................................................................... 325
Fields Tab ............................................................................. 325
Built In Fields ....................................................................... 325
The recCount Field .............................................................. 326
Creating New Fields ................................................................ 327
Creating New Fields via Import............................................ 327
Semantic Type ..................................................................... 327
Encrypted Fields ................................................................... 328
Secure Fields ....................................................................... 328
Code Sets ............................................................................ 329
Screens Tab ......................................................................... 330
Workflow Tab ....................................................................... 332
Data Store Node ................................................................ 332
State Node .......................................................................... 333
Transition ........................................................................... 334
Queue Node ......................................................................... 335
Queue Connection ............................................................... 336
Basic Workflow Example ..................................................... 336
Indexes Tab .......................................................................... 338
Query Filter Tab ................................................................. 338
Using a Filter Expression and a Super Group ..................... 339
Using a Filter Expression with User Profile Fields ............... 339
Using Multiple Query Filters .............................................. 340
Edit Restriction Tab ............................................................ 340
Other Tab ............................................................................. 341
Retention Example 1 ........................................................... 342
Retention Example 2 ........................................................... 343
Retention Period Over Time ................................................ 343
Retention Period Data Flush ............................................... 345
Building a Case Store for a Header/Detail Relationship:
The ‘Case contains record(s) from Data Store(s)’ Type .......... 347
Steps to Configure a ‘Case contains record(s) from Data
Store(s)’ Case Store ........................................................... 348
How to Check if Records already belong to a Case .......... 351
Building a Case Store for a Header Records Only:
The ‘Case is not associated with any Record or Stage’ Type... 354
Case Management ........................................................................ 354
Accessing Case Stores .......................................................... 355
Case Manager and Case Worker Roles ......................... 355
My Cases ................................................................................. 355
All Cases ................................................................................. 355
Dashboards ........................................................................... 356
Queues ................................................................................. 356
Search (Case Managers Only) ........................................... 356
Working Cases ......................................................................... 357
Viewing a Case ...................................................................... 357
Editing a Case ........................................................................ 357
Mass Editing Cases ............................................................... 357
Deleting a Case ................................................................. 357
Transitioning a Case ........................................................... 358
Quick Guide to Transitioning a Case ................................ 358
Merging a Case ................................................................. 359
Taking a Case ........................................................................ 359
Quick Guide to Taking a Case ............................................ 359
Fast Tracking a Case ......................................................... 360
Quick Guide to Fast Tracking a Case ................................. 360
Chapter 17  Data Entry

Searching the Data Store .......................................................... 365
Viewing Records ........................................................................ 365
Editing Records ......................................................................... 366
Deleting Records ....................................................................... 366
Creating Records....................................................................... 366
Unmasking Secure Fields.......................................................... 366

Chapter 18  General Product Functionality

Using Pipelines Across Multiple Instances .............................. 367
Exporting Definitions............................................................... 367
    What am I Exporting? ......................................................... 367
    What Should I Export? ..................................................... 367
    How do I Export Definitions? ............................................ 368
    Dirty State when Exporting Definitions .............................. 368
Promoting Pipelines Across Multiple Environments ............... 368
Navigating to a Selected Stage ................................................... 368
Finding Usages........................................................................ 369
Showing References ............................................................... 369
Testing an Expression in the Expression Editor ......................... 369

Chapter 19  Additional Help Material

Listing of Screencasts ............................................................. 371
Administrator Screencasts........................................................ 371
    Administrator Setup .......................................................... 371
    Delegating Rights to Data Stages ........................................ 371
    Custom Security .............................................................. 371
    Using the Executions History ............................................. 371
User Screencasts .................................................................... 371
Introducing Infogix Data3Sixty

What is Infogix Data3Sixty?

Welcome! If you’re reading this User Guide, your organization has decided to adopt a next generation self-service business intelligence platform designed to handle every step of your data processing work flow. Infogix Data3Sixty receives definitions and data from external sources and allows its users to create, share and analyze interactive data visualizations. Organizations that have chosen Infogix Data3Sixty have chosen to leverage the power of visual analytics on-demand and to apply that power to untamed sources of information already existent within their systems.

Infogix Data3Sixty System Overview

Infogix Data3Sixty was created to enhance business analysis, by allowing users to explore massive amounts of information, on-demand. With Infogix Data3Sixty, data exploration can take many different forms, allowing you to produce many different types of Data Stages.

Before diving into the specifics of creating different Data Stages, it can be helpful to understand where all of this takes place.

Exactly how Infogix Data3Sixty will be laid out for you will vary widely, depending on how your Administrator has configured the system. Nevertheless - and regardless of your role as a user - there are a number of components you should be familiar with.

Environments

In Infogix Data3Sixty, an Environment is the all encompassing “space” where a set of Data Stages exists. Typically, your Infogix Data3Sixty Administrator will create multiple Environments for multiple use cases, depending on what your organization needs. A typical setup might include Environments for Development, Testing, and Production. Each Environment can then be configured to contain only what its users need.
If you are granted access to multiple Environments, an Environments icon will appear in top right hand corner of your screen. Use this icon to switch from one Environment to another.

**Pipelines**

Within Environments, Data Stages are organized and associated in Pipelines. A Pipeline may contain any combination of Data Stages, and the contents/structure of any given Pipeline will be determined by your Infogix Data3Sixty Administrator, with the help of some users. Pipelines will typically contain an organization’s “curated content” - in the form of dashboards that serve as official reports and the multiple types of Data Stages underlying these dashboards.

**Creating Pipelines**

Users who help create Pipelines should be familiar with data management, transformation, and exploration. They’ll need to be able to create every type of Data Stage, and they can benefit from reading all of the content of this User Guide. If a deeper understanding of Pipeline structure and promotion across Environments is required, they may also benefit from reading the Infogix Data3Sixty Administrator Guide.

**Using Pipelines**

Other Users may only need to use Infogix Data3Sixty to view and interact with Data Stages created by other Users. For example, you may only need to view a Pipeline’s dashboards or use its Data Views to explore data. You may not need to be involved in the creation and maintenance of the Data Stages underlying reports. You may just want to perform self-service BI. If this sounds like you, you can safely skip Part One of this Guide, and read chapters dedicated to individual Data Stages as they are required.
Data Stages

Data stages are the objects which let you work with your data, within the Pipelines that exist in your organization’s Environments. Infogix Data3Sixty features a different Data Stage for each step of your data processing work flow.

**Data Store**

Data pushed to Infogix Data3Sixty resides in Data Stores.

Data Stores are containers for tabular information coming from internal or external data sources. Data Stores are essentially the starting point for all data exploration performed within Infogix Data3Sixty.

**Analysis**

An Analysis is essentially a drawing board for data manipulation. Analyses can be used to prepare, transform, and analyze data in preexisting Data Stores by applying functions to selected fields. These manipulations can then be output to other Data Stores.

**Analytic Model**

Analytic Models enable machine learning, by allowing you to Train, Score, and Evaluate data sets, via the Analytics nodes present in Analyses.

**Data View**

Data Views are used to define which fields within a chosen set of Data Stores can be used when creating Dashboards and exploring data in the Infogix Data3Sixty Visualizer.

**Process Model**

A Process Model is essentially a drawing board, used to orchestrate the flow of data loads and executions between interdependent Data Stages. The “output” of a Process Model is an automated work flow that can perform a number of tasks and ultimately keep your Data Stages in sync.
**Dashboard**

Dashboards are visual representations of Data Views, and they can display multiple pieces of information at once in many different ways. Upon completion, Dashboards can also be saved and shared for analysis and collaboration.

**Case Store**

A Case Store allows you to associate records from other Infogix Data3Sixty Data Stages in a common repository. These records can then be managed by users of the Case Store via screens that you define for the Case Store, to display specific information about cases.

**Rule Library**

A Rule Library allows you to create reusable rules that may be utilized within Analyses.
Working across Environments

As you create and work with different Data Stages, you may very well move through multiple Environments and Pipelines.

To become skilled at Infogix Data3Sixty data exploration, it can therefore be useful to begin with a high level consideration of how you will typically interact with the program. The following diagram provides a general process flow for a multi-Environment implementation of Infogix Data3Sixty, illustrating where incoming data comes from and the relationship between Data Stages.

Tip: As you view the following diagram, focus on the relationships between Data Stages rather than the specifics of Environment and Pipeline design. This diagram represents just one way Data Stages could be combined to process data. As you become more familiar with Infogix Data3Sixty, you will soon realize that there is practically an infinite amount of ways to structure a Pipeline; and, in practice your particular setup will be determined by your Administrator.
Incoming Data Pipeline

Development Environment

Testing Environment

Promotion Pipeline

Promotion Testing Environment

Promotion Production Environment
How will you use Infogix Data3Sixty?

This guide addresses the process of using Infogix Data3Sixty conceptually, assuming you are interacting with the program for the very first time. The presentation of chapters reflects the typical order in which you will need Infogix Data3Sixty’s features; however, chapters don’t necessarily need to be completed in any hard set order -- and depending on your role, you may not require all of the content in this guide.

Data Acquisition, Preparation, and Operationalization

In Infogix Data3Sixty, everything starts with Incoming Data. If you are familiar with your organization’s incoming data, you can use Infogix Data3Sixty to manage and prepare it. Sections of this guide dedicated to Acquisition, Preparation, and Operationalization include:

- Chapter 4: Data Store Configuration
- Chapter 6: Using the Analysis Designer
- Chapter 7: Creating Data Views
- Chapter 8: Executing Data Stages
- Chapter 9: Creating Process Models

Data Exploration and Analysis

Once incoming data has been acquired and prepared, you can begin to explore and analyze, creating ad-hoc data visualizations, Dashboards, or analytics using the Analysis Designer. Sections of this guide dedicated to Data Exploration and Analysis include:

- Chapter 6: Using the Analysis Designer
- Chapter 10: Building a Dashboard
- Chapter 11: Using the Infogix Data3Sixty Visualizer

Data Action: Sharing and Collaboration

After you, or someone else, has created and saved a dashboard, it will be available for collaboration and continuous use. Sections of this guide dedicated to these processes include:

- Chapter 13: Permissions and Collaboration Features

For more procedural guidance, Infogix Data3Sixty also includes step-by-step Guided Tours and Screen Casts.
Other Sources of Information

Your software includes documentation for each phase of implementation. The following table provides a complete list:

<table>
<thead>
<tr>
<th>Consult this document:</th>
<th>For this type of information:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator’s Guide</td>
<td>Conceptual overview of Infogix Data3Sixty administration topics.</td>
</tr>
<tr>
<td>User’s Guide</td>
<td>Conceptual overview of Infogix Data3Sixty user topics.</td>
</tr>
<tr>
<td>Screen Casts</td>
<td>Video tutorials covering key concepts. Accessible via Infogix Data3Sixty Help.</td>
</tr>
</tbody>
</table>

Contacting Customer Support

If you need assistance, contact Infogix Customer Support:

Call us at +1-630-505-1890 or submit a case online at support.infogix.com.

If you are experiencing issues contacting us by phone or by our website, please email support at support@infogix.com.

Part 1: Acquisition, Preparation, and Operationalization

Before Infogix Data3Sixty can be used to explore data, someone within your organization will need to acquire the data, prepare the data, and perhaps operationalize the data for continuous use. If that someone is you, you can proceed by reading Part One of this Guide. If someone else will be managing incoming data for you, simply skip this portion of the Guide and proceed to Part Two.

Infogix Data3Sixty can be used to prepare data in many ways. The diagram above depicts just one way you could use Data Stages to manage your organization’s incoming data.
Account Activation

1. **Register an Account**
   
   To log in and use Infogix Data3Sixty, you will need to register for an account. To do so, simply use the Register for an account link available on the log in screen.

   In addition to your personal credentials, you will need your organization’s Tenant ID. This is likely the name of your company, but if not your Administrator will be able to provide it for you.

2. **Click the Link in the Account Activation Email**

   After completing the registration form, you will receive a Welcome to Infogix Data3Sixty email containing a link. Click the link to activate your account, then log out of Infogix Data3Sixty and log back in. If you have trouble finding the Activation Email, check your Spam folder.

3. **Make sure that you are Claimed**

   If you are having trouble with the last part of Step 2 and find that you are unable to log into Infogix Data3Sixty, your account may not yet be claimed by your administrator. This can occur if you created your user account before your administrator created it in the Admin system.

   To resolve this issue, simply ask your Admin to Claim your account via the Users section of the Admin dropdown.

4. **Get Access to an Environment**

   The first time you successfully log in you may not have access to an Environment. Like 3) above, this could occur if your Administrator has not yet granted you access to an Environment. If this is the case, you will only have access to Infogix Data3Sixty’s Home Screen. To resolve this issue, ask your Administrator to add you to a Super Group and grant that Super Group access to an Environment. Once they have, you should be ready to go.
Logging in with Enterprise SSO

If your organization has configured Enterprise SSO, you can actually log into Infogix Data3Sixty without having to enter your user name and password - provided you are already logged into your organization’s network.

To do so, log in to your organization’s network and then navigate to Infogix Data3Sixty. Once you reach the log in screen, simply click the Enterprise SSO link and enter your organization’s Tenant ID.

Account Passwords

Changing Your Password

- On systems that are not using automated login via Enterprise SSO, users are required to change their passwords every 90 days. This is enforced at login time, if 90 days have passed since the last time a password was changed.
- When changing your password, it cannot be the same as your four previously used passwords.

Account Locking

- After 5 consecutive failed log in attempts, your account will be locked. To unlock your account, click the Forgot your password? link and follow the series of prompts that appear.

Why do I need a Secret Question?

- The answer to this question is used to help verify your identity when resetting your password.
Data Store Configuration

In Infogix Data3Sixty, Data Stores are containers for tabular information coming from internal or external data sources. Data Stores are essentially the starting point for all data exploration that will be performed, and for this reason they must be created and properly configured before Infogix Data3Sixty can be used.

Data Store Type

When creating Data Stores, you will first need to select the proper type to match your needs. Infogix Data3Sixty features two Data Store types: Internal and External.

Internal Data Stores

Internal Data Stores have multiple use cases:

1. Holding data pushed from another Application, such as Infogix Assure or Infogix ER.
2. Holding data coming from other Data Stores already set up within Infogix Data3Sixty. These types of Data Stores are typically outputs from other Data Stages - such as an Analysis that transforms a set of Data Stores to create a new one.
3. Holding the detail records for Case Stores, within a database.

Store Repository Types

Default
This is the option that will be chosen if you are pushing data from another Infogix application, or if you are creating a Data Store Output in an Analysis. (1 and 2)

Database
This is the option that will be chosen if you decide to create a Data Store that will be associated with and will hold detail records for a Case Store. To populate this type of Data Store, you will need to construct an Analysis that pushes data to it from other sources. (3)
4 Data Store Configuration

External Data Stores

This option includes an Indexes tab and a Screens tab that are not present in other Data Store types. These tabs are described below.

**Warning:** Internal Database Data Stores should only be used when you need to associate a Data Store to a Case Store!

External Data Stores

An External Data Store points to data coming into Infogix Data3Sixty from an external source, allowing for the exploration of virtually any set of data.

External data stores can be created in one of two ways:

1. Uploading a file directly.
2. Uploading data to an external location and then pointing to that location. This can take the form of uploading a file to some external repository and then pointing to that repository, or pointing to some preexisting database that contains your data.

Store Repository Types

For External Data Stores, ‘Store Repository Type’ refers to the external location at which your data is stored. Infogix Data3Sixty supports a number of different Store Repository Types. The following section provides an overview of the different types available within the different versions of Infogix Data3Sixty (Cloud or Enterprise). Implementation details for each type are covered in the Creating External Data Stores section that follows.

Infogix Data3Sixty Cloud Version

On the cloud, Infogix Data3Sixty currently offers two Store Repository Types for External Data Stores.

**Default**

This is the option that will be chosen if you decide to create an External Data Store via file upload. Behind the scenes, Default simply places your uploaded file into an Amazon S3 bucket. Each time you use the Default External Data Store, you are essentially accessing this bucket on S3.

**Database**

If you have a preexisting Database, this option allows you to point to it to access data from specific tables.
**S3**
This option also utilizes Amazon S3, however, instead of uploading a file via Infogix Data3Sixty, it requires you to upload files to an S3 bucket via AWS and then point Infogix Data3Sixty to that bucket. This option is preferable when you need to upload a large volume of data, because the Infogix Data3Sixty UI does not currently support multi-file upload.

**Infogix Data3Sixty Enterprise Edition**
If you are using Infogix Data3Sixty Enterprise Edition, you will have access to the following External Data Store Repository Types.

**Default**
In the Enterprise Edition of Infogix Data3Sixty, Default refers to the Hadoop Distributed File System, or HDFS. Here, choosing Default simply places your uploaded file into HDFS. Each time you use the Default External Data Store, you are essentially accessing HDFS.

**Database**
If you have a preexisting Database, this option allows you to point to it to access data from specific tables.

**File System**
If you have a preexisting file system other than HDFS, this option allows you to point to it to access data from specific files.

**Hadoop Distributed File System (HDFS)**
Like Default, this option also utilizes HDFS, however, instead of uploading a file via Infogix Data3Sixty, it requires you to upload files to HDFS and then point Infogix Data3Sixty to that location. This option is preferable when you need to upload a large volume of data, because the Infogix Data3Sixty UI does not currently support multi-file upload.

**Supported File Layouts**

**Delimited Layout**
On both the Cloud and Enterprise Edition, Data Stores can be creating using Delimited text files. Delimiters may be commas, tabs, or essentially any character you’d like. Whichever symbol is used, it is the Delimiter that tells Infogix Data3Sixty how to parse external data.
Common field delimiters such as Comma and Tab are listed in the delimiter dropdown; however, you should note that you are not limited to these options alone. You can actually enter any characters you’d like into the Delimiter text box, and if your file uses these characters to delimit fields Infogix Data3Sixty should properly parse the file.
Data Store Configuration

Supported File Layouts

Has Header Row
If your data file contains a header row, you can check this setting. Infogix Data3Sixty will then treat the first row in your file as a header and use this header to automatically name your fields.

Records Span Multiple Lines
This setting can be used when you have a field that contains line breaks, which causes records in the data set to span multiple lines.
For example, consider the following header (line 1) and record (lines 2-5) which spans multiple lines due to the presence of Line Feeds in the Description field.

Agency_ID, Agency_Name, Description, Zipcode, City, State
123, "Acme Labs", "Technical facility for testing big data concepts in LF the furthest reaches of space, time, and spreadsheets. Dedicated to LF unraveling the mysteries of the universe through data science. Also LF interested in interplanetary travel and data visualization.", "60555", "Warrenville", IL"

Checking the Records Span Multiple Lines check box would enable the system to process a file containing records such as this one. Leaving the setting off would likely result in an error.

Allow Lenient Parsing
Checking this box for a Data Store with a Delimited or Excel Layout will make it so that the structure of data loaded to the Data Store does not need to exactly match the structure defined in the Data Store’s definition.
Checking this check box will make it so that:

- Data files can contain more fields than are present in the Data Store definition.
  For data files that contain more fields than are present in the Data Store definition, when lenient parsing is set, an Overflow Field name parameter will become available when the Data Store is used as a Data Store Input node in an Analysis. Content from the extra fields will then be pushed into this Overflow Field in the JSON format when the Analysis runs.

- Data files can contain less fields than are present in the Data Store definition.
  For data files that contain less fields than are present in the Data Store definition, a Default Value field can be added to each Data Store field definition. When a file containing less fields than are present in the Data Store definition is used, the Default Value will be used to populate records in missing fields.

- Data files can contain fields in an order that is different than the order specified in the Data Store definition.

- Synonyms (described below) can be used for field names.
**Determining Delimiters**

This feature is used to automatically detect delimiters when accessing data that resides in an external location. If your file uses common delimiters such as Commas or Tabs, this feature will detect them. If other delimiters are used, they must be entered manually.

**Field Delimiters** are what separate your columns.

**Record Delimiters** are what separate your rows. These are usually not visible in your source file, but can be shown by modifying the settings of various text editors.

**Quote Character** should be specified if paired quotes are used around string fields.

For example, in the diagram above, specifying the Single Quote character prevents ‘bla,ck’ from being parsed into two fields due to the presence of a comma.

**Line Layout**

Data Stores can also be created using non-delimited files that are comprised of a single line.

**Steps to Generate a Line Layout Data Store**

1. Select a Store Repository Type. Files for Line Layout Data Stores can be uploaded to the Default Repository or accessed by pointing the Data Store to an external S3 location.
2. Specify a Line Layout. To parse a file into multiple records you can use the Line Delimiter Type.
   - **New Line** will parse the file into a new record whenever a CRLF or LF character is found*.
     For example, if your file read: ‘thisCRLFthat’
     Choosing the **New Line** option would parse the file into the following 2 records:
     ‘this’
     ‘that’
4 ■ Data Store Configuration

Supported File Layouts

- **Number of Characters** will take the specified number of characters from each CRLF or LF separated value in the file and use those characters to create a record. In other words, **Number of Characters** takes the first x characters from each line in the file.

  For example, if your file read: ‘thisCRLFthat’
  Setting Number of Characters to 3 would parse the file into the following 2 records:
  ‘thi’
  ‘tha’

3. Generate a field named ‘line’ in the Fields tab. You can either do this manually, or by using the Generate button.

4. If you chose the Default Repository type, you’ll now need to Upload a file to the Data Store.

   After creating the Line Layout Data Store, you can then see how your file has been parsed by using the Data Store in an Analysis.

   *Note: CRLF and LF characters are usually not visible characters in the file - they are line breaks. Here they are included to clarify the provided examples.

Additional File Layouts

In addition to Delimited text files and Line Layouts, both the Cloud and Enterprise Editions of Infogix Data3Sixty are able to support the following file layouts:

- **JSON** - limited to a single layer of nesting.

  For example:
  ```json
  {"name":"Michael", "age":45}
  {"name":"Andy", "age":30}
  {"name":"Justin", "age":19}
  ```

- **Parquet (Hadoop)**

- **ORC (Hadoop)**

- **AVRO (Hadoop)**

*Use traditional folder structure when used for output check box*

This check box allows you to output files to a traditional folder structure as opposed to a Hadoop folder structure, when using a Data Store as a Data Store Output in an Analysis. For Data Stores, this option is supported for External Data Stores with Delimited, Parquet, ORC, and AVRO layouts; and, it is available within the Details tab.

**Microsoft Excel Layout**

This layout option allows you to create Data Stores using Microsoft Excel files that have been saved in the .xlsx format. Using this layout type will allow you to create the Data Store using specific worksheets from the Excel file and to skip empty rows and columns if you need to.
**Skip Empty Columns**
Check this box if your worksheets contain empty columns that you would like to skip and not include in the Data Store.

**Skip Empty Rows**
Check this box if worksheets contain empty rows that you would like to skip and not include in the Data Store.

**Column Name Line Number**
This is the row number of the worksheets' header, which can be used to generate the field names of the Data Store.

**Identify Worksheets By**
As mentioned above, you can be selective about the worksheets within your Excel file used to create your Data Store. The following options may be used to do so.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>By Numbers</td>
<td>Specify worksheets using a comma delimited list of Worksheet indexes with no spaces, where indexing starts at 1. For example: 1,2,4</td>
</tr>
<tr>
<td>By Number range</td>
<td>Specify worksheets using a numeric range, where indexing starts at 1.</td>
</tr>
<tr>
<td>By Names</td>
<td>Specify worksheets by name, using a comma delimited list of Worksheet names. For example: august,september</td>
</tr>
<tr>
<td>By Name pattern</td>
<td>Specify worksheets using a regex pattern. All worksheets with names that match the regex pattern will be used.</td>
</tr>
</tbody>
</table>

**XML Layout**
This layout type allows you to create Data Stores using XML files. When this type is chosen, XML Layout parameters will become available to customize what contents are pulled from the file.

**Row Tag**
This should be the string that is used to tag rows within your XML file.
For example, consider the following XML structure:

```xml
<Roster>
```
Here, you could use Employee as your Row Tag. This would signify a structure where each Employee represents a row in your Data Store. The tags within that row could then be used as Data Store fields. Here, those fields would be RetailStoreID, WorkstationID, SequenceNumber, StartDate, EndDate, and OperatorID.

**Options**

Creating Options allows you to further specify how your XML file is read.

Options available for use include those listed in the read section here, except for `path`, which is handled by the product, and `rowTag`, which is handled by the parameter described above.

When using these options in the product, they must be prefixed with “read.xml”.

**Options Example**

Once again, consider the following XML structure.

```xml
<Roster>
  <Employee>
    <RetailStoreID>48</RetailStoreID>
    <WorkstationID>6</WorkstationID>
    <SequenceNumber>73</SequenceNumber>
    <StartDate>2014-09-30</StartDate>
    <EndDate>2015-09-30</EndDate>
    <OperatorID OperatorName="KERRY P">48237</OperatorID>
  </Employee>
</Roster>
```

In this example, the final field, OperatorID, contains an attribute, OperatorName. Using the excludeAttribute option, you could control whether this attribute was contained in the OperatorID field for this row.

To exclude the OperatorName attribute, you would create the following Option:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>read.xml.excludeAttribute</td>
<td>true</td>
</tr>
</tbody>
</table>
This would populate the OperatorID field with: 48237
To include the OperatorName attribute, you would create this Option instead:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>read.xml.excludeAttribute</td>
<td>false</td>
</tr>
</tbody>
</table>

This would populate the OperatorID field with: [KERRY P, 48237]

**Selection Criteria**
This parameter allows you to create SQL statements to selectively pull data from your XML file. When writing statements, ${ResultTable} should be used to refer to the table your XML file represents.

For example, consider the following XML structure:

```xml
<Roster>
  <Employee>
    <RetailStoreID>48</RetailStoreID>
    <WorkstationID>6</WorkstationID>
    <SequenceNumber>73</SequenceNumber>
    <StartDate>2014-09-30</StartDate>
    <EndDate>2015-09-30</EndDate>
    <OperatorID OperatorName="KERRY P">48237</OperatorID>
  </Employee>
</Roster>

<Roster>
  <Employee>
    <RetailStoreID>49</RetailStoreID>
    <WorkstationID>16</WorkstationID>
    <SequenceNumber>84</SequenceNumber>
    <StartDate>1999-07-23</StartDate>
    <EndDate>2006-01-04</EndDate>
    <OperatorID OperatorName="JOHN Q">39143</OperatorID>
  </Employee>
</Roster>
```

The following SQL statement would only pull JOHN Q into the Data Store:

```
SELECT * FROM ${ResultTable} WHERE RetailStoreID = 49
```
Creating Data Stores

To create a Data Store, your user account will need Create Data Store and Write permissions to a Pipeline.

The exact steps required to create and configure each type of Data Store described above vary by type.

Creating External Data Stores

External Data Stores are comprised of files stored on some external location. As mentioned above, there are multiple ways to create an External Data Store.

Direct Upload of File

To create an External Data Store via direct file upload, select Yes in the Create new data store from file dialogue and make the appropriate selections for delimitation. Uploading the file will load it to the Default Repository and automatically point your new External Data Store to it.

Create New Definition Version on each Save setting

If this setting is checked, a new version of your data store will be created in the back end file system each time you upload a new file, make changes to the Data Store’s configuration, and then Save.

S3, HDFS, or Alternative File System Upload

Alternatively, someone within your organization may have already loaded files to an external location, such as Amazon S3, HDFS, or some alternative File System. Once these files have been loaded, you can create a new Data Store and select No in the Create new data store from file dialogue. You’ll then need to configure the External Data Store to point to the appropriate channel.

Configuring Channel

These details point Infogix Data3Sixty toward the external data’s specific location. The exact parameters that are needed vary by Store Repository Type; however, you should be able to get all the information you need from the person who originally uploaded the files to that location.

Regular Expressions in File Paths

When configuring the external channel, either literal strings that represent the appropriate file path or regular expressions that represent the file path may be used.
Filter Extension
This setting can be used when you have files in the same location that share the same name but have a different file extension. The specified Filter Extension will tell Infogix Data3Sixty which file to use to create the Data Store. For example, if you had a folder containing a `customerData.txt` and a `customerData.csv`, specifying either `.txt` or `.csv` as a Filter Extension would allow you to choose between the two.

Marker Extension
This setting can be used to prevent writing to files that should not be written to unless a Marker is found. It is available for the S3 and HDFS file systems.

S3, HDFS, or Alternative File System for Analysis Output
If you would like to output the results of an Analysis to external, file based Data Store, you may do so. You can either pre-create the Data Store and then point to it using a Data Store Output node in the Analysis, or you can create it on the fly within the Analysis.

Outputting Executions to Unique Folders
When using external, file based Data Stores as Analysis outputs, you can set things up so that a new folder that will be used to contain outputted files is created upon each execution. Folder names can be generated using the following set of variables:
- `workId` (the work id of the current execution)
- `refStartTime` (in yyyy/MM/dd HH:mm:ss.SSS format)
- `refStartTimeYear`
- `refStartTimeMonth`
- `refStartTimeDate`
- `refEndTime` (in yyyy/MM/dd HH:mm:ss.SSS format)
- `refEndTimeYear`
- `refEndTimeMonth`
- `refEndTimeDate`
- `now` (in yyyy/MM/dd HH:mm:ss.SSS format)
- `year`
- `month`
- `date`

Variables should be used within the external, file based Data Store’s path parameter, using the `{$variable}` syntax.
For example:

`folder1/folder2/${now}`
4 Data Store Configuration

Creating Data Stores

Each time the Analysis that output to this Data Store ran, it would create a new folder within the external file system. The folder’s name would be the current time stamp. The folder’s contents would be the files from the output of the Analysis. The next time the Analysis ran, it would then create another new folder, named using the current time stamp and containing the files from the output of that run.

Setting up Analysis output files in this fashion can be a good way to organize and keep a history of the files that were created during specific Analysis executions.

Custom Delimiters on Outputs
When using an external Data Store as the output of an Analysis, you can set the Data Store’s Delimiters to any characters you’d like. After running an Analysis that pushed to this Data Store, you can then download the files that comprise the Data Store’s contents and they will be delimited as you’ve specified.

Database
If you have a preexisting database, you can also use an External Data Store to point to it and pull data from specific tables.

Configuring Channel
These details allow Infogix Data3Sixty to connect to your database. You should be able to get all of the channel information you need from the person managing the database.

- **Database Type** refers to the brand of database you are using. If your type is not listed, you can use the Other type.
- **JDBC Driver Class** refers to the component that helps Infogix Data3Sixty communicate with your database. Common drivers for each Database Type will autofill this parameter when a Database Type is selected.
- **URL** refers to the external location of your database.
- **Username** and **Password** refer to the credentials you use to access the database.

Testing your connection to the Database
To test your connection to the database, you can use the Test button. Testing should always be performed prior to Field Generation to ensure that communication with the database has been properly established.

DB2 Database Support
While not explicitly listed, you can point to a DB2 database by selecting the Other Database Type option.

Note: As of release 2.1.1, DB2 based Data Stores may not be used as Data Store Outputs within an Analysis; that is, Infogix Data3Sixty 2.1.1 cannot write to DB2 based Data Stores.
Data Store Configuration  ■  4

Creating Data Stores

**SQL Layout**
This parameter is where you can write an SQL query to pull data from the tables you’d like included in your Data Store. During Field Generation, each field pulled from the database becomes a field in the Data Store.

For example, the statement:

```
SELECT ID, CREATETIME FROM student_Infogix Data3Sixty_data_pull
```

Would allow you to generate ID and CREATETIME fields in your Data Store.

Whereas:

```
SELECT * FROM student_Infogix Data3Sixty_data_pull
```

Would allow you to create all the fields from student_Infogix Data3Sixty_data_pull in your Data Store.

As with standard SQL, more advanced statements may also be made. For example, you could use a WHERE condition to pull in specific records for specific fields, or a JOIN to combine fields from multiple tables.

**SQL Statement versus Sample SQL Statement**
Within the SQL Layout, there is an SQL option and a Sample SQL option. Both fields allow you to construct an SQL statement. The statement written in the SQL field is the statement that will be used to create the Data Store definition, whereas the statement written in the Sample SQL field is a statement that will be used when sampling the Data Store in an Analysis.

For example, consider the following statements:

- **SQL:** `SELECT * FROM EMPLOYEE where ID > 2`
- **Sample SQL:** `SELECT * FROM EMPLOYEE where ID > 5`

Due to the SQL statement, a Data Store that used these statements would bring in all records from the EMPLOYEE table that had an ID greater than 2. These records would be available for processing throughout the application. Due to the Sample SQL statement, however, Analysis sheets that displayed this Data Store’s contents would only display records with IDs greater than 5.

Note that the use of a Sample SQL statement is optional. If a Sample SQL statement is not used, Analysis sampling will occur as it does with other Data Stores, according to Analysis settings.

**Concurrent Reads**
This setting can be used to optimize the speed at which data is read from a Data Store’s database, particularly when the database contains a large number of records.
4 ■ Data Store Configuration

Creating Data Stores

For example, if the database contained 1,000,000 records, a normal query would utilize 1 read, to read all 1,000,000 records. If you were to increase the number of Concurrent Reads to 10, however, this would cause the query to use 10 reads in parallel - and each read would handle 100,000 records. Making this setting could potentially increase the speed at which the Data Store queries its database.

To experiment with different numbers of Concurrent Reads, you can use the User Defined setting and simply choose a number.

Alternatively, you can let the system make this optimization for you, by selecting the System Defined option.

Using Order By in your SQL Query with Concurrent Reads
If you have selected System Defined or User Defined for your Concurrent Reads parameter, you will also need to order your SQL Query by a field that contains unique values for each record in the data set. Note however that you will not need to do so with your Sample SQL Query.

Output Table Name
If the Database Data Store is going to be used as a Data Store Output within an Analysis, you need to specify the name of the table within the external database that the Data Store is using.

Generating Fields for a Database Data Store
Once you have configured your SQL Channel and Layout, you can go to the Data Store’s Fields tab and use the Generate button to generate fields. Assuming you have successfully connected to the database, generated fields and the records that populate them will be based on the SQL you have written in your layout.

Using substitutable time stamp variables in SQL statements
The following substitutable variables may be used in your SQL:

<table>
<thead>
<tr>
<th>Variable name</th>
<th>Timezone applied</th>
</tr>
</thead>
<tbody>
<tr>
<td>fromTimestamp/toTimestamp</td>
<td>Applies the timezone of the system’s web application server to the time value.</td>
</tr>
<tr>
<td>fromTimestampUTC/toTimestampUTC</td>
<td>Applies the UTC timezone to the time value.</td>
</tr>
<tr>
<td>fromTimestampOrgTz/toTimestampOrgTz</td>
<td>Applies the timezone set in the Admin-&gt;System screen of the product to the time value.</td>
</tr>
</tbody>
</table>
Creating Internal Data Stores - External Applications

Creating an Internal Data Store that will hold data pushed from another Application consists of a few basic steps, the completion of which may require assistance from your Infogix Data3Sixty Administrator.

1. **Connecting to Infogix Data3Sixty**
   To bring data into Infogix Data3Sixty from another Application, your administrator will need to create an “Application User” profile. This profile will allow the other product to communicate with Infogix Data3Sixty.

2. **Push Definitions to Create Data Stores**
   Once a connection is established, you can push the definitions from your external application to Infogix Data3Sixty.

   If pushing from Infogix Assure or Infogix ER, doing so will automatically create Data Stores in Infogix Data3Sixty for you. These Data Stores will be located in an Incoming Pipeline, containing Paths based on the controls that have been configured in Assure or ER.

   Note: If you intend to associate an Internal Database Data Store from Infogix Assure to a Case Store in Infogix Data3Sixty, it will be labeled as a ‘Case Output’ Data Store within the Incoming Pipeline in Infogix Data3Sixty.

3. **Selecting and Moving Data Stores**
   To use your newly created Data Stores, you will need to Move them out of the Incoming Pipeline and into another Pipeline/Path. Moving the Data Stores lets the external application know that it should push data to those Data Stores the next time an execution occurs.

4. **Execute the entity in the External Application to push data to the Data Store**
   Once you have moved a Data Store from the Incoming Pipeline to another Pipeline, this will signal that data should be pushed to this Data Store the next time the associated entity is executed within an external application. Only Data Stores that have been moved out of the Incoming Pipeline will have data pushed to them. All the Data Stores residing in the Incoming Pipeline are effectively empty definitions until they are moved.

   Note: It is important that you Move the Data Store out of the Incoming Pipeline; do not just Copy it. In order to Move a Data Stage, you need Write permission to the destination Pipeline/Path.

   Once moved into a new Pipeline/Path, you can then configure the Data Store’s properties and use it to create other Data Stages.
Creating Internal Data Stores - Data Stage Outputs

If you are planning to create an Analysis, you can create an Internal Data Store to serve as the Analysis’ output ahead of time. Otherwise, you can simply create this type of Internal Data Store while you are designing the Analysis.

Creating Internal Database Data Stores - for use with Case Stores

As mentioned above, if you want to associate a Data Store with a Case Store - having it contain detail records that support the header records of a Case Store - you will need to create an Internal Database Data Store. After it is created, you can populate it by pushing data into it via an Analysis; and, you can associate it to a Case Store by adding it to the Case Store’s Workflow.

Warning: Internal Database Data Stores should only be used when you need to associate a Data Store to a Case Store!

Creating Fields for an Internal Database Data Store

Prior to pushing data to an Internal Database Data Store via an Analysis, all of the fields that the Internal Database Data Store will receive need to be created.

- If the Internal Database Data Store is created on-the-fly, within the Analysis, these fields will be created automatically.
- If you want to create the Internal Database Data Store ahead of time, you can create all of the required fields by using the Import Fields feature, and importing from the source Data Store.

System generated fields when an Internal Database Data Store is associated to a Case Store

When an Internal Database Data Store is added to a Case Store’s Workflow, it will become “owned” by the Case Store. Within the Data Store, this ownership is represented by the addition of the following set of system fields:

- **OwningObjectId**
  The system generated id of the Data Stage that “owns” Data Store.

- **OwningObjectType**
  The type of the Data Stage that “owns” Data Store.

- **OwningSubObjectUID**
  The UID of the case record from the Case Store that owns the detail record in the Data Store. As detailed in the Case Stores section of this guide, this UID must be generated and tied to header/detail records in an Analysis using the UUID() function.

- **OwningSubObjectID**
  Rather than the UID of a case record that owns the detail record in the Data Store, this field contains the ID of a case record that owns the detail record in the Data Store. This ID is system generated.
Data Store Configuration

Creating Data Stores

Creating Data Stores that allow Ad Hoc Queries

Within a Data Store’s Detail tab, there is a check box called *Allow Ad Hoc Queries.* Checking this checkbox will allow the Data Store to be used in data visualizations, without the need to set up a Data View. While convenient, this setting should only be used to gain a basic understanding of a data set while building an Analysis. Attempting to create a production level dashboard on an Ad Hoc Query Data Store will result in poor performance.

In addition, the following should be considered:

- The Ad Hoc Query setting is only available for file based Data Stores. Specifically: Internal S3/HDFS, External S3/HDFS, and External Default for Delimited, AVRO, Parquet, and ORC layouts. Note that the setting is checked but disabled for Internal Database Data Stores, because databases are inherently able to be queried.
- External, non-default Data Stores of type HDFS or S3 will allow you to specify partition fields that will be used to partition the data when performing ad hoc queries. This parameter is available in the Data Store’s Other tab.
- For Internal S3 and External Default Data Stores on the cloud version of the product, the Ad Hoc Query setting is only available for Data Stores that were created in version 3.0 of the product or later.
- For Internal HDFS, External HDFS, and External Default Data Stores on the enterprise edition of the product, the Ad Hoc query setting is only supported if Apache Hive is configured.
- For Internal S3/HDFS and External Default Data Stores, ad hoc queries are not supported if any field names in the Data Store clash with one of the following reserved field names: file-id, work-id, month, date, year, version, datastore-name, devenv-id, owning-member-id.
- For External Data Stores using the Delimited layout, ad hoc queries are only supported if the record delimiter is Newline or Carriage Return Newline.
- The *Cache Query Results* checkbox that appears below the Allow Ad Hoc Queries checkbox can be checked to improve performance. If this setting is checked, duplicate queries will use cached data. This checkbox is enabled and turned off by default for Internal Database Data Stores. It is disabled and turned on for Internal s3/HDFS, External s3/HDFS, and External Default Data Stores for the layout types that support ad hoc querying. It is disabled and turned off for all other Data Stores.

SysDeleted

A Boolean flag that indicates whether a record has been deleted and can be physically deleted from the system at any time.
Data Store Fields

This property allows you to create Data Store fields and modify existing ones.

Field Types

Big Integer
Integer values where a Precision greater than 19 is needed.

Boolean
Meant for the values True and False.

Currency
Numeric values with currency signs. For example, $1.99 or €1.50.
Like the Decimal type, Currency will give you complete control over rounding by enabling the Rounding Type and Scale options.

Date
Meant for values that represent some combination of a year, month, and day, and no hour-minute-second time stamp. Values with timestamps will have all times set to 00:00:00 if the Date data type and a Format Pattern without a timestamp is applied.

DateTime
Meant for values that represent some combination of a year, month, day, hour, second, and time zone.

Time
Meant for values that represent some combination of hour, minute, second, period (AM or PM), and time zone.

Decimal
Meant for non-integer numeric values where complete control over rounding for values to the right of the decimal point is needed. Applying the Decimal data type will give you the ability to specify a Rounding Type and Scale. This type should be applied when slight changes to values due to computational rounding are not tolerable.

Floating Point
Data Store Configuration

Data Store Fields

Meant for non-integer numeric values where control over rounding for values to the right of the decimal point is *not needed*. Can be applied when slight changes to values due to computational rounding are tolerable.

**Integer**

Integer values with a maximum **Precision** of 19.

**String**

Meant for text values. For example, a person’s name or the name of a product.

**Tip:** Applying a data type to data that wouldn’t seem to fit that type can sometimes allow you to cast values into new types, which could enable the use of other functions in other Data Stages. For example, applying the String data type to Integers would allow you to use functions designed for Strings on these values in an Analysis.

**Semantic Type**

Applying a Semantic Type to any type of field can help to further categorize the field’s content and enable additional functionality throughout Infogix Data3Sixty.

For example, adding the Airport semantic type to a string field containing airport codes could help to categorize a dataset containing unlabeled airport codes within an Analysis.

*Displaying URLs as Hyperlinks in Dashboards with Semantic Type*

As another specific example, the URL Semantic Type can be applied to String fields that contain String values which should be treated as URLs. Doing so will actually cause the URL field’s Strings to be displayed as clickable hyperlinks in Data Grid dashlets. If the URLs are valid, clicking on these links will open the associated website in another browser tab.

**Default Value**

For data files that contain less fields than are present in the Data Store definition, a Default Value field can be added to each Data Store field definition. When a file containing less fields than are present in the Data Store definition is used, the Default Value will be used to populate records in missing fields.

**Settings for Numeric Fields**
Format Pattern
This option allows you to specify how numbers are formatted for processing. The best way to determine whether you need to explicitly define the Format Pattern is to use the Data Store Field Test feature. If the Test fails at a numeric field, you most likely need to define a Format Pattern for that field. If the Test passes, you can simply leave Format Pattern blank.

For more help with Format Patterning, see the Number Format Java Doc.

Display Format Pattern
This setting determines how numbers will be displayed in the Data Store UI and in Analysis Sheets. If left blank, Display Format pattern will default to Format Pattern settings.

Precision
This setting controls the number of digits to include in an integer value. This will affect integers in other Data Stages that use the field as well, such as Analyses, Data Views, and Dashboards.

If you have customized Precision for a Data Store field and you use that Data Store field in a Data View, you should set the Data View field’s Precision to match that of the Data Store’s, to avoid execution errors.

If you intend to use a field in a Data View, it is also recommended that you do not use Precision as a means of removing digits in a Data Store because this will lead to errors in Data View execution. For more on this, see the Precision section of Data View Fields.

If Precision is left blank, it will default to 19, which is also the maximum value that Precision can be set to.

Scale
This setting controls the number of digits to the right of the decimal point that will be included in numeric values. For example, if value = 1.234 and Scale is set to 2, value will equal 1.23 within the Data Store.

Additionally, setting Scale higher than the amount of digits to the right of the decimal point that are present will add zeros. For example, if value = 1.234 and Scale is set to 4, value will be displayed as 1.2340.

If Scale is left blank, the field will pick up all digits to the right of the decimal point.

Scale when a Field is Edited In-Line or Used in a View/Edit Screen
In the special cases of a numeric field being used within Case Management and being edited in-line or being displayed in a View/Edit Screen, the following rules apply:

- Decimal fields respect the Scale setting specified for the field or will have a decimal precision of 4 if no Scale is specified.
Data Store Configuration

Data Store Fields

- Currency fields respect the Scale setting specified for the field or will have a decimal precision of 2 if no Scale is specified.
- Floating point fields will have a decimal precision of 4.

Rounding Type

Round Up

The Round Up option rounds up the last decimal digit before the cutoff specified in Scale, if the cutoff digit is greater than or equal to 5.

For example if \textit{value} = 1.05, then choosing Round Up and setting Scale to 1 rounds \textit{value} to 1.1. If \textit{value} = 1.04, however, the same settings would round \textit{value} to 1.0.

Truncate

Truncating a number will cut off all values that are to the right of the decimal point and beyond the number’s Scale.

For example, if \textit{value} = 1.245, then choosing Truncate and setting Scale to 2 would cause \textit{value} to equal 1.24. If on the other hand Scale were set to 1, \textit{value} would equal 1.2.

Currency Code

For Data Stores, Currency fields are predominantly configured through the Format Pattern option. Specifying a Currency Code is not required for Currency fields; however, if one is specified it will be passed on to any Data View that uses the Data Store Currency field.

Specifying a Currency Code in Data Store can therefore save you an extra step down the line, if you know that the field will be used in a Data View. This is because Currency Codes are required to display currency signs in Dashboards and the Visualizer.

To make more sense of this, consider the following example, using a Data Store Currency field called \textit{amt}.

Example: Formatting Currency Fields

If \textit{amt} = $100, you would first need to select the US currency format in the Format Pattern option (\$#,##0.00;-$#,##0.00) to make $100 a parsable value. If this was the only thing you did, your Data Store would pass Testing and be usable in Analyses - however the dollar sign ($) would not appear in Analysis sheets.

To have the dollar sign ($) displayed in the Data Store UI or Analysis Sheets, you would also need to specify the US currency format as the field’s Display Format Pattern.
To have the dollar sign ($) displayed in Dashboards or Visualizations, you could then specify USD as the field’s Currency Code in the Data Store, so that the setting will be passed with the field when the field is used in a Data View. Alternatively, you could leave Currency Code blank in the Data Store and then just specify USD in your Data View later.

**Settings for Date, DateTime, and Time Fields**

*Time Zone*

This setting is used to define a Time Zone for values in Date, DateTime, and Time fields. It should be used to apply a Time Zone to values that do not already have a Time Zone within their Format Pattern.

Specifying a Time Zone will alter values in the Data Store and in other Data Stages that use the Data Store’s field. This Time Zone setting will also interact with the System Time Zone that is specified in Infogix Data3Sixty Administrator Preferences.

The following examples illustrate how this interaction plays out, using a Data Store field called `dateTime`.

**Time Zone: Example 1**

If `dateTime = 2016/02/02 12:00:00` and the field’s Time Zone option is set to +0100, then `dateTime` becomes `2016/02/02 11:00:00` in a system using GMT time.

Specifying +0100 means that the original value in `dateTime` is one hour ahead of GMT time, so the system, which is set to GMT time, displays this value as `2016/02/02 11:00:00` in GMT, instead of `2016/02/02 12:00:00`.

**Time Zone: Example 2**

If the System Time Zone is different than GMT, this setting will interact with the Data Store field Time Zone.

For example, if `dateTime = 2016/02/02 12:00:00`, and field Time Zone is set to +0100, and System Time Zone is set to +0200, then `dateTime` becomes `2016/02/02 13:00:00`.

Again, specifying +0100 means that the original value in `dateTime` is one hour ahead of GMT time (12:00:00 - 1:00:00 becomes 11:00:00 GMT) but since the System is set to two hours ahead of GMT time, `dateTime` becomes `2016/02/02 13:00:00` (11:00:00 + 2:00:00 = 13:00:00).
Time Zone: Example 3

As mentioned above, if there is already a Time Zone in a field’s value and Format Pattern has been configured to parse it, selections in the field Time Zone dropdown will be ignored.

For example, if `dateTime = 2016/02/02 12:00:00 -0100`, and Format Pattern is set to `dd/MM/yyyy HH:mm:ss Z`, and the System is using GMT time, then `dateTime` will become `2016/02/02 13:00:00` regardless of any setting made in the field Time Zone dropdown.

In this situation, a System Time Zone other than GMT will affect `dateTime` as well.

For example, if `dateTime = 2016/02/02 12:00:00 -0100`, and Format Pattern is set to `dd/MM/yyyy HH:mm:ss Z`, and the system is using `-0200` time, `dateTime` will become `2016/02/02 11:00:00` regardless of any setting made in the field Time Zone dropdown.

Here, the `-0100` in the original `dateTime` value means that `dateTime` is one hour behind GMT (12:00:00 + 1:00:00 becomes 13:00:00 GMT), but since the system is set to two hours behind GMT, `dateTime` becomes `2016/02/02 11:00:00` (13:00:00 - 2:00:00 = 11:00:00).

Settings for String Fields

Max Length

This setting controls the maximum number of characters allowed in values for String fields. Max Length will be set to 255 by default.

If you have customized Max Length for a Data Store field and you use that Data Store field in a Data View, you should set the Data View field’s Max Length to match that of the Data Store’s, to avoid execution errors.

For more on this topic, see the Max Length section of Data View fields.

Generating Fields

When creating a new data store, Fields can be automatically generated through use of the Generate button.

If generating on an Internal Data Store that has been pushed to Infogix Data3Sixty from another application, fields will be created as they exist in the source application. If generating on an External Data Store, Infogix Data3Sixty will use the specified Delimited Layout settings to generate the fields.

Importing Fields

For ease of use, the Import Fields feature can be used to create fields by pulling their definitions from another Data Store. Imported Fields will have matching names, field types, and all other configurations.
4 Data Store Configuration

Data Store Fields

Testing Fields
Once Fields have been generated, you should use the Test button to ensure that the Field values are being properly read. If there are issues with a specific Field, the test will display details that can help you fix the problem.

After generating Fields, you should always Test until you’ve ran a successful test to make sure the Data Store will work.

Showing Sample Data
After a successful test, you may use the Show Sample Data option to see a sample of the data in your data store. Note that this sample is just a subset of your data store’s contents.

Setting up a Single Field Data Store for Parsing
Data stores that have a single field that needs to be parsed in an Analysis should be configured as follows:

- Set the Field Delimiter in the Details tab to a blank entry, to specify no delimiter
- Set the Field Name in the Fields tab to line

Since it is also common for records to be parsed to be very long Strings, the Field’s Max Length attribute may also need to be increased.

Encrypted Fields
Specifying a Date Store field to be encrypted will cause the field’s records to be encrypted in storage.

Encryption Limitations
Only Internal Data Stores may have encrypted fields.

Permissions Required to Modify Encryption
- Any user with access to a Data Store can encrypt a Data Store field if it hasn’t yet been encrypted.
- Once a field has been encrypted, only Administrators and Users with Administer or Manage Security permission to the Data Store will be able to modify the field’s encryption settings.

Inheritance when used by other Data Stages
- If a field is encrypted, this will carry over when the field is used in other Data Stages.
  - In the case of a Data View, the encryption will be inherited and will not be able to be changed by anyone - unless it is changed in the Data Store.
Places where you’ll see the effects of Encryption

- When you use the Data Store in an Analysis and view its contents in a Sheet. Here, encrypted values will be masked - even when they are not secured. Also note that you will not be able to unmask encrypted values within an Analysis.

Effects on Data when Changing Field Encryption

- If changing field encryption on an Internal Default Data Store, any preexisting data in the Data Store will not be encrypted/decrypted.
- If changing field encryption on an Internal Database Data Store, a migration job will be performed upon save to encrypt/decrypt the Data Store's pre-existing data.

Secure Fields

Specifying a Data Store field to be a Secure Field will cause the field to be masked throughout Infogix Data3Sixty. Masking is defined through use of the parameters described below.

Secure Field Parameters

- **Hide this field when user does not have access**
  Users who have access to a Secure Field are those who belong to the Super Groups chosen in the Secure Field’s Super Groups parameter. If this box is checked and a user who is not part of the Secure Field’s Super Groups uses the Data Store, they will not see the Secure Field at all.
  If this box is left unchecked, all users will see the Secure Field, but its contents will always be masked.

- **Mask**
  This setting determines how the Secure Field will be masked.

- **Mask Pattern**
  If the User Defined Mask option is chosen, you can use Mask Pattern to define a preset or custom mask for the content of your Secure Field.

- **Display Mask Pattern**
  If the User Defined Mask option is chosen, you can use Display Mask Pattern to define a preset or custom display mask for the content of your Secure Field.

- **Super Groups**
  Users in a Secure Field’s selected Super Groups will be able to see the field with its contents initially masked, as per Mask settings. Unlike other users, however, users within the selected Super Groups have the option to unmask Secure Fields and see their actual contents. Unmasking can be performed throughout the product, at the individual record level or in bulk via Unmask All.
  If a user is not in a Secure Field's selected Super Groups, they will either see the field as always masked or they won't see the field at all. This will depend on whether ‘Hide this field when user does not have access’ has been checked.
4 ▬ Data Store Configuration

Data Store Fields

Permissions Required to Create or Modify Data Store Secure Fields
- Any user with access to a Data Store can make a Data Store field Secure, if it hasn’t yet been marked as secure.
- Once a field has been marked as Secure and the change has been Accepted, only Administrators and Users with Administer or Manage Security permission to the Data Store will be able to modify the field’s security settings.

Inheritance when used by other Data Stages
- If a field is set as Secure, all the security settings will carry over when the field is used in other Data Stages.
  - In the case of a Data View, the Secure flag’s setting will be inherited and will not be able to be changed by anyone - unless it is changed in the Data Store.
  - When Data View field security is inherited from a Data Store, Infogix Data3Sixty Administrators and users with Administer or Manage Security permission to the Data View will be able to modify settings besides the Secure flag. Other users will not be able to modify any of these settings.

Places where you’ll see the effects of Secure Fields
- When you download the contents of a Data Store and view the file outside of Infogix Data3Sixty
- When you use the Data Store in an Analysis and view its contents in a Sheet
- When viewing a Secure Field within a Data Store Search Screen. Here, you will also be given the option to unmask the value of a masked Secure Field, if you have been given permission by being placed in the Secure Field’s Super Group.
- Dashboards and the Visualizer. Here, some functionality will give you the option to unmask the value of a masked Secure Field, if you have been given permission by being placed in the Secure Field’s Super Group.

Code Sets
A Code Set may be used to create a more human readable display value for fields that contain non-human readable codes. Once a Code Set is created, its effects will be seen throughout the system - such as when the field’s values are displayed in a Search Screen or a Dashboard.

Display Name
This is the value that will replace the raw value that is actually stored in the Data Store field, when the field is displayed throughout the system.

Value
This is the raw value that will be replaced by the display name. Typically, it will be a non-human readable code.
**Code Set Example**
Assuming a field that contains a set of 3 non-human readable codes, you could create a Code Set for each one, in order to produce a more user-friendly display.

**Data Set with Obscure Codes**

<table>
<thead>
<tr>
<th>id</th>
<th>obscureCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>GHTY45RD</td>
</tr>
<tr>
<td>002</td>
<td>JJAK09WQ</td>
</tr>
<tr>
<td>003</td>
<td>QWRE21PP</td>
</tr>
</tbody>
</table>

Assuming you created the following:
Display Name: Dental, Value: GHTY45RD
Display Name: Medical, Value: JJAK09WQ
Display Name: Home, Value: QWRE21PP
‘Dental’, ‘Medical’, and ‘Home’ would be displayed for GHTY45RD, JJAK09WQ, and QWRE21PP (respectively) whenever these values were encountered.

**Synonyms**
When creating a Data Store field, Synonyms allow you to add words to the field definition that will also be interpreted as the field’s name, when the word is encountered as a field name in a data source.

For example, if you had some files where an employeeId field was represented as `employeeId` and other files where employeeId was represented as `id`, you could name the Data Store field `employeeId` and add `id` as a synonym.

**Example File 1**

<table>
<thead>
<tr>
<th>employeeId</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
</tr>
<tr>
<td>002</td>
</tr>
<tr>
<td>003</td>
</tr>
</tbody>
</table>
Example File 2

With id added as a synonym, source data with an employeeId column or an id column would be treated as if it belonged to the same column. In this example, this common column would be employeeId, since that is the name defined for the Data Store field. As a result, data would reside in the Data Store as follows:

Data in Data Store

<table>
<thead>
<tr>
<th>employeeId</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
</tr>
<tr>
<td>002</td>
</tr>
<tr>
<td>003</td>
</tr>
<tr>
<td>004</td>
</tr>
<tr>
<td>005</td>
</tr>
<tr>
<td>006</td>
</tr>
</tbody>
</table>

Transform

When working with Data Stores, the Transform tab allows you to perform custom transformations on your source data using scripts. Like other places in Infogix Data3Sixty - notably, the Script nodes available in Analyses and Process Models - Transform tab scripts are written using JavaScript. With Data Stores, one useful application of the Transform tab’s scripting is the parsing of a single field into multiple, new fields.

Note: The Transform tab is not currently available for Internal type Data Stores; however, similar functionality can be achieved by using an Internal Data Store in an Analysis.

Script Inputs

To work with Data Store fields within your script, you must first add the fields as Input Fields, using the Add Fields button. Once Input Fields have been selected, you can then refer to them in your script as input.fieldName.
**Input Fields**
Specifying a field as an Input Field makes the Data Store field available for manipulation in the Script.

**New Field**
Creating a New Field creates a field that can be used in your Script, to hold parsed data as it is passed from Input Field to Output Field.

As a very simple example, consider parsing an Input Field named 'line', which you'd like to output into an Output Field named 'line'. If you created a New Field named 'temp', you could do something like this in your script (assuming all fields were Strings):

```javascript
input.temp = input.line.slice(0,2);
output.line = input.temp + " that is a slice";
```

Doing so would create records in output.line where each record contained the first two characters of each line in 'line' with “ that is a slice” concatenated.

**Script Outputs**
Script Outputs are the fields that have been created in the Fields tab of your Data Store. They are the containers that will hold the outputs of your Script.

To use an Output Field in your script, you can refer to it as `output.fieldName`.

**Preserve Source Order**
Checking this box will cause the incoming records to remain in the same order as they appear in the data source. If left unchecked, records will be parsed by the script in random order.

**Script Controls Emit**
Check this box if you want to manually control which records from your data source are emitted by the script - typically via use of the emitRecord() function. Leaving this box unchecked will cause all records from your source to be emitted by the script.

**Using the Script Input and Script Output tabs to handle Javascript’s Numeric Limitations**
When using Javascript it is important to be aware of the language’s limitations, particularly when dealing with numeric fields.

- The maximum numeric integer that can be reliably used in Javascript is 9007199254740991. Any number greater than this may potentially be truncated when using Javascript.
4 ▪ Data Store Configuration

Identity

To avoid truncation of field values greater than 9007199254740991, you will need to tell the Transform script to treat a field’s values as Strings. To do so, use the Script Input and Script Output tabs to mark the field as a String type. If you would still like to treat the field as an Integer or Big Integer in the Data Store, you can mark the field as such in the Data Store’s Fields tab.

Script Rules tab

Here, you can incorporate preexisting Script Rules from Rule Libraries into your Transform Script. Doing so will allow you to call functions from the incorporated rules by name, within your Script. To incorporate a Script Rule, you just need to identify its parent Rule Library and Rule Group, and then select the Script Rule you want to use.

Identity

This property allows you to select one or more identity fields for the Data Store.

Identity Fields for Data Stores Associates to Case Stores

When creating an Internal Database Data Store that will be associated to a Case Store, it is required that you specify a set of Identity Fields for the Data Store and that you check the box labeled Identity fields are unique.

Also note that when you specify a field as a unique identity field, it actually needs to have a unique value for each record - otherwise executions using this Data Store will fail.

Indexes (Internal Database Data Store only)

If the search functionality provided in a Data Store’s Search Screen(s) is found to be slow, you can attempt to improve it by creating an Index. Index creation simply involves selecting a field from the Data Store that is used to perform searches.

To gauge whether the Index has improved the Data Store’s search, you should then attempt to search for records using the same field that was used to create the Index.

Automatically Generated Indexes

Once an Internal Database Data Store is associated to a Case Store, the following indexes will be automatically created within the Data Store.

- IDX_OwningSubObjectID
- IDX_OwningSubObjectUID
- IDX_SysDeleted
As with manually created indexes, these indexes are intended to improve search speed when searching the contents of an Internal Database Data Store. Within the UI, you can also: change the fields that comprise these indexes, change the order in which the fields that comprise the index are used, and specify whether to sort the fields that comprise the index in ascending of descending order.

Screens (Internal Database Data Store only)

The Screens Tab is where you can build the UI components that allow users to interact with records within an Internal Database Data Store. Two types of screens can be created: Search Screens and View/Edit Screens.

Search Screens
A Search Screen is what a user will use to search for a specific record. When creating a Search Screen, you get to define the criteria that will be used to search for specific records by way of: Filter Fields and Result Fields. You will also need to define how search results appear by specifying a Result View Screen Name and a Result Edit Screen Name.

For Internal Database Data Stores, Search screens are utilized when:

- Users search for detail records in a Case Store
- Users drill to an Internal Database Data Store from a Dashboard

Filter Fields
Filter Fields are the fields from the Data Store with which you want users to search for records. Selected fields will appear within the search screen interface with the specified Label and Width. Additionally, the chosen Component Type will determine how the user can search, where available Component Types depend on the data type of the chosen field.

Wildcard Search for String Filters
String fields used in filters specifically support both literal and wildcard search. Wildcard search can be performed through use of the % character.

Result Fields
Selected Result Fields are those that will appear for records returned by a search using Filter Fields. Result Fields will appear as specified by the Label and Width parameters.

Specifying a Result Field as Editable
Specifying a Result Field as Editable will enable inline editing within Data Entry. This allows the user to double click on the cell value of a result field and change it inline, without having to go to the record’s Edit screen.
Data Store Configuration

Query Filter (Internal Database Data Store only)

Sort Fields
Sort Fields are fields from the Data Store to use for sorting the results of a search. Only fields that are also used as Result Fields may be used as Sort Fields. Additionally, a Sort Field may be specified in Ascending or Descending order, using the Ascending check box.

Create Screen Name
In order to specify a Create Screen Name for a Search Screen, you must first have created a View/Edit Screen (detailed below). This selection will determine the display of the form used to create new records via Data Entry in the Data Store.

Result View Screen Name
In order to specify a Result View Screen Name for a Search Screen, you must first have created a View/Edit Screen (detailed below). This selection will determine the display of a record that a user has searched for and then decided to view.

Result Edit Screen Name
In order to specify a Result Edit Screen Name for a Search Screen, you must first have created a View/Edit Screen (detailed below). This selection will determine the display of a record that a user has searched for and then decided to edit.

View/Edit Screens
View/Edit Screens determine how a record is displayed after a user has found it with a Search Screen and then decided to view/edit it. View/Edit Screens essentially allow you to create a form that represents a record, which the user may interact with. The form will be comprised of fields from the record, which may be editable or non-editable, based on your requirements.

Screen Layout
When creating a View/Edit Screen, Screen Layout is used to determine the basic arrangement of form fields in the screen.

Field Properties
Once a Screen Layout is created, selecting each field will reveal a Field Properties panel. This panel allows you to define which fields should appear at what points within your layout and how user interaction with the field should occur.

Query Filter (Internal Database Data Store only)
Query filters allow you to restrict which data from an Internal Database Data Store certain users will be able to access. They can be written to apply a Filter Expression to Super Groups of users who query the Data Store and only allow those users to see certain records.
When combined with User Profile fields they can also be used to filter queried content at a user level, with no Super Group assignment required. Examples of both types of Query Filter are shown below.

**Using a Filter Expression and a Super Group**

As an example, consider a data set that contains a state field containing values from all 50 United States (IL, CA, OK, etc.). Also suppose that a system Super Group named ‘Midwest’ already exists, containing users who are only supposed to work with data from the Midwest U.S. region.

To construct a Query Filter that would only allow users in the Midwest Super Group to see data from Midwest states, we could write the following Filter Expression:

\[
\text{state} = \text{‘IL’} \mid\mid \text{state} = \text{‘IA’} \mid\mid \text{state} = \text{‘WI’}
\]

(Note that more states could be added if they needed to be considered part of the Midwest U.S. region)

We would then need to select the Midwest Super Group as the Super Group to apply this Query Filter to. By default, this Query Filter would then only allow users in the Midwest Super Group to see data from IL, IA, and WI. Importantly, users not in the Midwest Super Group would see nothing.

**Allow user to view all data when there are query filters but no matching query filter is found for a user Checkbox**

In the section above it is noted that once a Query Filter with a Super Group assignment is applied, the default behavior is that users who are not part of that Super Group will see no data when they query the Data Store. Checking the **Allow user to view all data...** checkbox will reverse this behavior, allowing users who are not part of the Query Filter’s Super Group to see all data in the Data Store.

**Using a Filter Expression with User Profile Fields**

As an alternative to applying Query Filters at a Super Group level, Query Filters can be combined with User Profile fields to apply restrictions at an individual user level.

As an example, consider a scenario where the following User Profile fields have already been set up on the following user accounts:

<table>
<thead>
<tr>
<th>User Account</th>
<th>Profile Field Name</th>
<th>Profile Field Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>state</td>
<td>IL, MA</td>
</tr>
<tr>
<td>Jane</td>
<td>state</td>
<td>CA</td>
</tr>
</tbody>
</table>

In this scenario, a Query Filter with the following Filter Expression could be used:
**Data Store Configuration**

*Edit Restriction (Internal Database Data Store only)*

dataStoreStateField = PROFILE_FIELD('state')

Here, 'state' refers to the Profile Field name set on the user account and dataStoreStateField refers to a state field in the Data Store containing values from all 50 U.S. states. When a user queries the Data Store, the Query Filter will then only show them records where one of their Profile Field values matches the dataStoreStateField value. With this type of set up, no Super Group assignment is required.

**Using Multiple Query Filters**

When multiple Query Filters are applied, the Query Filters are treated in an OR-like fashion. That is, each Query Filter is considered independently.

For example, if you had the following expressions being used as Query Filters:
1. field1 > 0
2. field2 < 100

Values such as -1 or 101 would both make it through the Query Filters.

To get an AND effect, you would need to combine these expressions into a single Query Filter.

**Edit Restriction (Internal Database Data Store only)**

Edit Restrictions allow you to restrict which data from an Internal Database Data Store certain users will be able to edit in Data Entry. They apply an Expression to Super Groups of users and only allow those users to edit certain records. Like Query Filters, Edit Restrictions may also alternatively utilize User Profile Fields.

**Edit Restriction Example**

By default, when no Edit Restrictions are applied to an Internal Database Data Store, all users with Read Data and Change Data permission to the Data Store will be able to edit all of its records, using the Edit Screens that have been designed for the Data Store. Adding an Edit Restriction will make it so that users in a selected Super Group have more limited access to what they can edit.

For example, consider a situation where there is a status field in a Data Store, containing values such as Open, Closed, In Review, etc., and that we only want users to be able to edit records that have an Open status. Additionally, suppose that there already exists a Super Group of users named 'Editors' to which we would like to apply this restriction.

To do so, we could create an Edit Restriction with the following Expression and apply it to the Editors Super Group:

status = ‘Open’
By default, this Edit Restriction would make it so that whenever users in the Editors Super Group performed Data Entry on the Data Store, they would only be able to edit records with an Open status. Users not belonging to the Editors Super Group would not be able to edit anything.

*Allow user to edit all data when there are edit filters but no matching edit filter is found for a user Checkbox*

In the section above it is noted that once an Edit Restriction with a Super Group assignment is applied, the default behavior is that users who are not part of that Super Group will not be able to edit any records in the Data Store. Checking the *Allow user to edit all data...* checkbox will reverse this behavior, allowing users who are not part of the Edit Restriction’s Super Group to edit all data in the Data Store.

**Using an Edit Restriction with User Profile Fields**

Alternatively, User Profile Fields may be used with Edit Restrictions. This type of set up works similarly as in Query Filters, and the example described above for Query Filters could also be applied to an Edit Restriction. The only difference would be that the Profile Field look up would control what records users could edit, not which records they could view.

**Links**

This property allows you to link fields from multiple data stores. Multiple data stores should be linked if you intend to use them in a single data view.

**Profile tab**

A Data Store’s Profile tab allows you to profile the Data Store’s data without having to create and run an Analysis. Choosing to Profile a Data Store will make it so that you can Execute the Data Store. Such an execution will push the profiling information to the Data Stores you specify within the Profile tab. This information can then be viewed in summary within a *Metrics* tab that will appear on the profiled Data Store after profiling is complete and in detail by performing ad-hoc visualization or analysis on the created Profile Data Stores.
**Data Store Configuration**

*Profile tab*

**Profiling Parameters**

*Data Load Range*

This parameter allows you to define the range of data within the Data Store that will be profiled.

*Data Load Range Options*

<table>
<thead>
<tr>
<th>Option</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Profiling will read all data from the Data Store, every time profiling occurs.</td>
</tr>
<tr>
<td>New Data Since Last Load</td>
<td>Profiling will only read data that is new, since the last time the Data Store loaded. If no new data is found, no changes will be made to the Profiling outputs.</td>
</tr>
<tr>
<td>Based on Date Parameters</td>
<td>Profiling will only read data from a specified date range.</td>
</tr>
<tr>
<td>Based on Work ID Parameter</td>
<td>Profiling will only read data from a specified Work ID.</td>
</tr>
<tr>
<td>Based on File Path Pattern</td>
<td>Profiling will only read data from files with a file path that matches the pattern parameter.</td>
</tr>
</tbody>
</table>

*Sampling Rate Percent*

This parameter allows you to specify what percentage of records from the Data Store should be profiled. Note that this is always an approximation.

*Value Limit for Field Values*

The Value Limit for Field Values parameter controls how many values can be displayed for a field in a Profile Values Data Store or a Profile Patterns Data Store that is created when profiling. If more values are found in a field than the Value Limit, no values will be displayed for that field. By default, Value Limit is set to 10,000.

*Custom Counters*

Custom Counters allow you to create expressions to apply to individual fields, which are used to create a count of values that satisfy the expression.

For example, consider the following data set.
Custom Counter Data Set

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>100</td>
</tr>
<tr>
<td>002</td>
<td>125</td>
</tr>
<tr>
<td>003</td>
<td>150</td>
</tr>
<tr>
<td>004</td>
<td>175</td>
</tr>
<tr>
<td>005</td>
<td>200</td>
</tr>
</tbody>
</table>

Were you to create a Custom Counter field named `greaterThan150` using the expression `value < 150`, `greaterThan150` would equal 2.

Adding Custom Counters to the Data Store Profile tab will add an integer field for each Custom Counter to the Profile Data Store. These field names will match the names given to the Custom Counters.

Profile Data Store

This parameter is required when Profiling a Data Store, as it specifies which Data Store the profiling information should be pushed to. Here, you can point to a preexisting Data Store or use the Create button to create a new one. Using Create will create a new Internal Database Data Store with the following fields:

- `dataStoreId` (String), `field` (String), `totalCount` (Integer), `distinctCount` (Integer), `distinctPercent` (Decimal), `uniqueCount` (Integer), `uniquePercent` (Decimal), `duplicateCount` (Integer), `duplicatePercent` (Decimal), `nullCount` (Integer), `emptyCount` (Integer), `blankCount` (Integer), `completePercent` (Decimal), `max` (String), `min` (String), `maxLength` (Integer), `minLength` (Integer).

In addition, if you have set up Custom Counters in the Profile tab, an integer field is created using the counter name for each counter. If you have set up Data Quality Checks in the Profile tab, an integer field is created using the Result Count Field parameter as the name for each rule defined.

The identity fields for this Data Store are `dataStoreId` and `field`.

Data Quality Checks

Adding a Data Quality Check to the Profile tab of a Data Store allows you to run reusable rules from a Rule Library on the Data Store’s data. When Data Quality Checks are added to profiling, the Data Store’s Profile Data Store will gain a new field for each added check. Such fields will contain a count of how many records return true for the Data Quality Check and will use the Check’s Result Count Field as their name. If Data Quality Checks are added, you can also push data that represents the results of running the checks to a Profile Rules Data Store.
4  Data Store Configuration

Profile tab

Adding a Data Quality Check

- Rule Library
  This is a Rule Library that you have already created, which contains the reusable rules you want to apply to your Data Store during profiling.

- Rule Group
  Within the selected Rule Library, this is the Data Quality Rule Group that contains the reusable rules that you want to execute.

- Result Field
  This parameter allows you to give a name to the result field that will be produced by the Data Quality Check in the Profile Rules Data Store. This field will contain a Boolean result. The result for each evaluated record - True or False - is the result of ANDing every reusable rule applied to the record. In other words, to produce a True result, an evaluated record must cause every reusable rule within the Data Quality Check to evaluate to True.

- Error Reason Field
  This parameter allows you to give a name to the error field that will be produced by the Data Quality Check in the Profile Rules Data Store. If a record’s Result Field evaluates to False, the Error Reason field will contain the list of reusable rules which evaluated to False. If a record’s Result Field evaluates to True, the Error Reason field will be empty.

- Execute All Rules in a Group
  Check this box if you would like to apply every reusable rule from the selected Data Quality Rule Group to the Data Store during profiling.

- Rules to Execute
  If you have unchecked Execute All Rules in a Group, you can select individual reusable rules from the Data Quality Rule Group to use.

Placeholder Mapping tab

The Placeholder Mapping tab is used to map incoming fields to the Placeholder Fields within the selected Data Quality Rule Group. Incoming fields that are mapped to Placeholder Fields will be evaluated in the selected reusable rules wherever the Placeholder Fields are referenced, in the rules’ definitions.

Profile Patterns Data Store

This parameter allows you to pick a Data Store to store the profile pattern information that is generated when profiling. Here, you can point to a preexisting Data Store or use the Create button to create a new one. Using Create will create a new Internal Database Data Store with the following fields:

dataStoreId (String), field (String), pattern (String), patternRegex (String), totalCount (Integer), patternPercent (Decimal).
Data Store Configuration

**Profile tab**

The identity fields for this Data Store are dataStoreId and field.

**Profile Values Data Store**
This parameter allows you to pick a Data Store to store the profile values information that is generated when profiling. Here, you can point to a preexisting Data Store or use the Create button to create a new one. Using Create will create a new Internal Database Data Store with the following fields:
- dataStoreId (String), field (String), value (String), totalCount (Integer), valuePercent (Decimal).

The identity fields for this Data Store are dataStoreId and field.

**Profile Rules Data Store**
This parameter allows you to pick a Data Store to store the results of the execution of Data Quality Checks set up in the Profile tab. Here, you can point to a preexisting Data Store or use the Create button to create a new one. Using Create will create a new Internal Database Data Store with the following fields:
- dataStoreId (String), result field (Boolean) and error reason field (String) for each Data Quality Check defined.

The identity fields for this Data Store are dataStoreId and the result fields.

Note that if the Create button is used, the system will create an Internal Database Data Store of the Parquet layout type. If you are pointing to a predefined Internal Database Data Store, however, any layout type can be used.

**Executing a Data Store for Profiling**
Once you have set up a Data Store’s Profile tab, it will become executable - and, executing the Data Store will cause the profiling to be performed. Once the execution is complete, the Data Store will gain a Metrics tab, which will contain a summary of each field’s profiling information. This summary comes from the profiled Data Store’s Profile Data Store.

Additionally, each Profile Data Store that was set up in the Profile tab will be populated with profiling information. You can then view the profiling information using ad-hoc queries in the Data3Sixty DQ+ Visualizer or in an Analysis. You can also view a summary of the information contained within the Profile Values and Profile Patterns Data Stores by viewing the profiled Data Store’s Metrics tab and using the View Values and View Patterns buttons.
Other tab

Data Retention

Retention Period defines the length of time for which a Data Store will store data. Data Retention applied at the Data Store level will override Retention Period settings at the Environment level made by your Administrator.

Period Precision defines the precision at which a Retention Period is applied.

For example, if a Data Store’s Retention Period was set to 1 year and its Period Precision was set to Month, the Retention Period would span one year back from the current day, plus the remaining days to get to the beginning of the month.

Retention Example 1

For instance, suppose the current day is 6/5/2016.

If Data Store Retention Period was set to 1 year and Period Precision was set to Month, the system would first look back one year to 6/5/2015. Then, since Precision is set to Month, the system would go to the beginning of the month, to 6/1/2015. At this point, any data that was loaded into the Data Store before 6/1/2015 would be removed.

Retention Example 2

As another example, suppose the current day is 6/5/2016.

If Environment Retention Period was set to 1 year and Period Precision was set to Quarter, the system would first look back one year to 6/5/2015. Then, since Precision is set to Quarter, the system would go to the beginning of the quarter, to 4/1/2015. At this point, any data that was loaded into the Data Store before 4/1/2015 would be removed.
Retention Period Over Time

Data Retention is applied once per day, and is calculated from the end of the most recent successful load or execution. If a run fails, it will evaluate from the failed run’s start time.

As time passes, retention is always evaluated from the end of most recent successful load or execution. If a run fails, it will evaluate from the failed run’s start time.

Data Retention is applied once per day, and is calculated from the end of the most recent successful load. If a load fails, Data Retention will be calculated from the failed load’s start time.
4 ■ Data Store Configuration

Other tab

Retention Period Data Flush

Over time, data will be flushed from Data Stores based on the Data Retention settings specified. For instance, in the example above, data from Load 1 on 4/29/2015 is flushed on 5/1/2016. This is because a 1 year retention period with Month precision is used; so, on 5/1/2016, the 4/29/2015 Load falls outside of the calculated Retention period (which spans from 5/1/2015 to 5/1/2016).

Additional Data Store Retention Processing Tasks
During retention processing, any records in an Internal Database based Data Store that have a SysDeleted field value of true will be deleted from the Data Store.

Data Retention Not Available on External Data Stores
Due to the fact that Infogix Data3Sixty cannot delete the contents of an external location, Data Retention is not permitted with non-default External Data Stores.

Data Retention is permitted on External Data Stores of the Default type and Internal Data Stores.
Partition By
This option is available for External (non-default) Data Stores of type HDFS or S3. For such Data Stores, if the partition fields are specified, then the organization of files in the Data Store will be in the Hadoop format with folders of the form <fieldname>=<fieldvalue> for each of the partition fields, in the order they are specified in the UI. If no partition fields are specified, the files should be directly under the path specified for the Data Store.

Handling Data Store Contents

Using the Upload File feature
After an External Data Store has been created and saved, you can continue to upload files to it.
Assuming a file has the same field structure as the file that was used to create the External Data Store, an upload will simply add new data to the Data Store.
If, on the other hand, you attempt to upload a file with a different field structure than the file that was used to create the Data Store, errors may occur.

Executing Dependent Data Stages After Upload
When manually uploading files to a preexisting Data Store, it is important to consider how the upload will affect other Data Stages that use that Data Store. After upload, you therefore may need to Run or Rebuild executable Data Stages connected to the Data Store.
Depending on your permissions, and your familiarity with the Pipeline you’re working with, such executions could be: carried out manually, scheduled, or orchestrated via Process Models.

Using the Delete All Data Feature
When working with an External Data Store in the Edit Stage context, you can use the Delete All Data feature to remove all data that is currently loaded to the store.
The need to Delete All Data from a Data Store may arise:
- In between Analysis Runs or Rebuilds
- In cases where you’d like to upload a new file with the same field structure to an external data store and are not interested in the data contained in the old files.

Affect on other Data Stages
Note that Deleting All Data can have an impact on other Data Stages that use the Data store. To determine if this will be an issue, simply use the Find Usages feature on the Data Store while viewing it in a Pipeline.
Handling Data Store Contents

When Dependent Data Stages are Executing
Deleting all Data from a Data Store while Data Stages that use that Data Store are executing will cause the execution to fail. For this reason, you should always check to see which Data Stages use the Data Store before deleting its Data. If you need help with this, contact your Infogix Data3Sixty Administrator.

Viewing Data Store Contents
To view Data Store Contents, use the View Stage option available from your Pipeline. Alternatively, you can click the View button present in the toolbar when editing the Data Store.

Viewing a Data Store allows you to take a look at the files that were used to create it, or its “Contents.” With Data Stores, these Contents can also be downloaded for local use. When viewing contents, files are grouped by Data Store load events - and Data Store load events are labeled by system generated Work IDs.

When viewing contents, you can also reach each load event’s execution history directly, by selecting the Work ID you are interested in and then using the View Execution button available in the toolbar.

Permissions
Permission needed to View: Read
Permission needed to Download files: Read Data
Permission needed to view Load Event Execution History: Administer

File Download Formats
- **Internal Data Stores and External Data Stores created from File**
  Files within these Data Store types are available in the compressed .bz2 format. These files must be extracted before you view them on your local machine.
Rule Library

This Data Stage allows you to create reusable, data quality rule sets that can be applied to data in an Analysis. Reusable rules created with a Rule Library are used in an Analysis via the Execute Rule Library node.

Details tab
Within Details tab, you may give your Rule Library a Display Name and a Description. This information can help you find the Rule Library later, when you want to use it in an Analysis, via an Execute Rule Library node.

Rules tab
The Rules tab is where you can create the different types of reusable rules, for use in an Analysis. Different types of rules are created with the New button, and each type of rule is discussed below.

Data Quality Rule Group
A Data Quality Rule Group is a top level folder, used to hold reusable rules. When building your reusable rules, you must first create at least one Data Quality Rule Group.

Preferred Result Field
When creating a Data Quality Rule Group, the Preferred Result Field is the parameter used to name the field that will hold the result of a reusable rule within an Analysis.

Preferred Error Reason Field
When creating a Data Quality Rule Group, the Preferred Error Reason Field is the parameter used to name the field that will hold errors resulting from the evaluation of a reusable rule within an Analysis.

Placeholder Fields
Placeholder Fields enable the ‘reusability’ of a Rule Library, by allowing you to define what types of fields the reusable rules contained within the Data Quality Rule Group can accept. Placeholder Fields can be given virtually any name, however their data types should match the data types of the fields you intend to evaluate with your rules.
Completeness Rule

A Completeness Rule is used to determine whether a selected field has completely passed a set of checks. If all checks pass, True is returned; if at least one check fails, False is returned.

Placeholder Field

This is a field from the Data Quality Rule Group that you would like to use in the Completeness Rule. When using the Completeness Rule in an Analysis, the data type of the field being evaluated in the Analysis will need to match that of the Completeness Rule’s Placeholder Field.

Not Null

This check returns True if a record value found in the Placeholder Field is not null, or False if a record value found in the Placeholder Field is null.

Not Empty

This check returns True if a record value found in the Placeholder Field is not null or has a String length greater than 0, or False if a record value found in the Placeholder Field is null or has a String length of 0.

Not Blank

This check returns True if a record value found in the Placeholder Field is not blank, or False if a record value found in the Placeholder Field is blank. Values that are considered blank are those that are empty (as defined above in Not Empty) or those that contain only spaces.

Static Text

Used to compare values in the Placeholder field to a static String. If a record value found in the Placeholder Field matches the Static Text, this check returns True. If a record value found in the Placeholder Field does not match the Static text, this check returns False.

Check Expression

This parameter allows you to create an expression using functions and the Completeness Rule’s Placeholder Field. If a record value found in the Placeholder Field causes the expression to evaluate to True, this check will return True; otherwise, this check will return False.

Consistency Rule

A Consistency Rule is used to evaluate whether a record’s values satisfy an expression. If a record satisfies the conditions in an expression, the Consistency Rule will return True. Conversely, when a record doesn’t satisfy an expression, the node will output False.
Note: There is no Placeholder Field parameter for Consistency Rules because all Placeholder Fields from the Data Quality Rule Group may be used.

*Check Expression*
This parameter allows you to create an expression using functions and the Consistency Rule’s Placeholder Field. If a record value found in the Placeholder Field causes the expression to evaluate to True, this check will return True; otherwise, this check will return False.

*Value Conformity Rule*
A Value Conformity Rule is used to check whether data set values match specific values or fall within a range of values. If a data set value matches any of the values in a Value Conformity Check or falls within the Check’s range, a True value is returned. Conversely, if a data set value does not match any of the values in a Value Conformity Check or falls outside of the Check’s range, a False value is returned.

To reverse the functionality, you can select 'Negate results of checks' to create an 'exclusion list'. In this case, the rule checks that specific values, or a range of values, are excluded from a data set. With this option selected, if a data set value matches any of the values in a Value Conformity Check, or falls within the specified range, a value of False is returned. If there are no matching values, or if the data set values fall outside of a the Check’s range, then a value of True is returned.

*Placeholder Field*
This is a field from the Data Quality Rule Group that you would like to use in the Value Conformity Rule. When using the Value Conformity Rule in an Analysis, the data type of the field being evaluated in the Analysis will need to match that of the Value Conformity Rule’s Placeholder Field.

*Checks*
- **Value List**
  Checks to see if a value matches any values from a set of values. Values in the set may be added one at a time using the New Value option or as a comma separated list with the Multiple Values option.

- **External Value List**
  Also checks to see if a value matches any values from a set of values, however here the set of values should come from a Data Store field within your Pipelines. The specific Data Store to be used is selected via the *Source Data Store* parameter. The specific field within the Data Store to be used is selected via the *Values Field* parameter. Once these parameters are specified, unique values from the Data Store’s field will be used for rule evaluation.
Type Conformity Rule

A Type Conformity Rule is used to check whether data set values conform to patterns. If a data set value matches the pattern in a Type Conformity Check, a True value is returned. Conversely, if a data set value does not match the pattern in a Type Conformity Check, a False value is returned.

Placeholder Field

This is a field from the Data Quality Rule Group that you would like to use in the Type Conformity Rule. When using the Type Conformity Rule in an Analysis, the data type of the field being evaluated in the Analysis will need to match that of the Type Conformity Rule’s Placeholder Field.

Checks

- **Regular Expression**
  Checks to see if a value conforms with the specified Regular Expression Pattern.

- **Number**
  Checks to see if a value conforms to a specific number formatting pattern. Either predefined formatting patterns or a custom pattern may be used.

- **Date**
  Checks to see if a value conforms to a specific date formatting pattern. Either predefined formatting patterns or a custom pattern may be used.

Regular Expression Rule

The Regular Expression Rule is used to check whether values in a Referenced Placeholder Field conform to a specific Regular Expression.

Referenced Placeholder Field

This is a field from the Data Quality Rule Group that you would like to use in the Regular Expression Rule. When using the Regular Expression Rule in an Analysis, the data type of the field being evaluated in the Analysis will need to match that of the Regular Expression Rule’s Referenced Placeholder Field.

Regular Expression Pattern

Checks to see if a value conforms with the specified Regular Expression Pattern.

Timeliness Rule

The Timeliness Rule is used to check whether the amount of time between two fields surpasses an acceptable level. If the difference between the two chosen fields is less than or equal to the acceptable interval, the record is
considered timely and a *True* value is returned. Conversely, if the difference between the two chosen fields is greater than the acceptable interval, the record is not considered timely and a *False* value is returned.

**Start Time Placeholder**
This is a Date, Date and Time, or Time field from the Data Quality Rule Group that you would like to compare to another, end time field.

**End Time Placeholder**
This is a Date, Date and Time, or Time field from the Data Quality Rule Group that you would like to compare to another, start time field.

**Allowed Time Difference**
If the difference between the two chosen fields is less than or equal to the Allowed Time Difference, the record is considered timely and a *True* value is returned. Conversely, if the difference between the two chosen fields is greater than the Allowed Time Difference, the record is not considered timely and a *False* value is returned.

---

**General Expression Rule**
A General Expression Rule is used to determine whether values in a Referenced Placeholder Field cause a Check Expression to evaluate to True or False.

**Referenced Placeholder Field**
This is a field from the Data Quality Rule Group that you would like to evaluate in the General Expression Rule. When using the General Expression Rule in an Analysis, the data type of the field being evaluated in the Analysis will need to match that of the General Expression Rule’s Referenced Placeholder Field.

**Check Expression**
This is the expression that is used to evaluate the rule’s Referenced Placeholder Field. When it is used in an Analysis, it will evaluate to either True or False, depending on the expression and the values it encounters.

---

**Script Rule Group**
A Script Rule Group is a top level folder, used to hold reusable Script rules. When building your reusable rules, you must first create at least one Script Rule Group.

---

**Script Rule**
A Script Rule is used to create a reusable script in Javascript. This rule can be reused wherever scripting can be performed, such as in a Data Store’s Transform tab or within an Analysis Script node.
Creating the Script and Function Signatures
When building a Script Rule, the Edit Script button can be used to write functions in Javascript. For each function you create, you should also create a Function Signature that defines a Name, Description, Return Data Type, and Parameters for the function.

Basic Script Rule Example
Suppose that within a Script Rule, you create the following Script:

```javascript
function addOne(var1) {
  return var1 + 1;
}
```

With the following Function Signature:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Return Data Type</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>addOne</td>
<td>Adds 1 to value</td>
<td>STRING</td>
<td>var 1</td>
</tr>
</tbody>
</table>

You could then use this Script Rule within an Analysis Script node. To do so, you would just need to add the Rule Library, Script Group, and Script Rule that contained the addOne function to the Script node using the node’s Script Rules tab. Once you have done this, you should be able to call the addOne function in the Script node’s script by name.

For example:

```javascript
output.field1 = addOne(input.field1);
```

Assuming input.field1 contained the following data:

```
field1
1
2
3
```

output.field1 would contain the following after the script had run:

```
field1
2
3
4
```
Using a Rule Library in an Analysis

Once you have created a Rule Library containing a Data Quality Rule Group and some reusable rules, you may use the Rule Library in an Analysis via the Execute Rule Library node. With this node, usage is simply a matter of selecting which Rule Library and rules to execute and mapping the node’s incoming fields to the placeholder fields defined in the Rule Library.

For more detailed information on the Execute Rule Library node, see the Using the Analysis Designer section of this guide.
Using the Analysis Designer

Once you have a Data Store containing incoming data, you can create an Analysis.

A Infogix Data3Sixty Analysis is essentially a drawing board for data manipulation. Analyses can be used to prepare, transform, and analyze data in preexisting data stores by applying functions to selected fields.

The output of an Analysis is typically a new data store containing prepared data that is better organized and thus better suited to serve as an input for other Data Stages.

For example, if you only wanted to visualize on certain data store fields from multiple incoming data stores, you could use an analysis to create a single data store comprised of those fields. You could then use that new data store to create a data view.

Creating an Analysis

As the remainder of this chapter will describe, there are numerous ways an Analysis can manipulate data. Accordingly, there are also numerous ways to go about building an Analysis.

The general process flow of any Analysis is as follows:
In the first phase of design, Data Store Inputs are selected (1). In the second phase, fields from these data stores are manipulated, as they move through some combination of Enhance, Combine, Shape, Check, System, and Analytics nodes (2). In the third phase, manipulated fields are pushed to and stored in one or more Data Store Outputs, which are then saved in a Pipeline to be used by other Data Stages.

**Prerequisites**
To create an Analysis, your user account will need Create Analysis and Write permissions to a Pipeline.
You will also need access to at least one data store.
In terms of prerequisite knowledge, users with varying levels of familiarity with their organization’s incoming data can benefit from using the Analysis Designer.
Those with prior knowledge of a data set may be able to approach design with a specific output in mind. On the other hand, the Analysis Designer can also work as a tool for experimentation and exploration, no prior knowledge required.

**Saving an Analysis**
This option will save the Analysis definition and the sample data associated with each node.

**Saving an Analysis Definition**
This option allows you to save only the Analysis definition, and not the sampled data associated with each node. Saving only the definition and not the sampled data can be useful if you’d like to reduce load time when editing the Analysis again.

**Analysis Designer Nodes**
The Analysis Designer contains a number of drag and drop nodes used to manipulate data. Below is an overview of each node and its unique properties.

**Input/Output Nodes**

**Data Store Input**
Data Store Input nodes are usually the starting points of every Analysis. They are used to select which data stores you would like to manipulate.
Selecting a Data store
To select the data store to use as an input, use the Data Store drop down in the Properties panel. Available stores will be those you have been granted access to, in Pipelines.

Selecting a Data Load Range
When selecting a Data Store Input, it is important to specify a Data Store Load Range. This setting will define which data is read by the Analysis, and ultimately which data is placed in Data Store Outputs when the Analysis executes.
### Data Load Range Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Effect</th>
<th>Useful when:</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Analysis will read all data from the Data Store Input, every time the Analysis Runs.</td>
<td>Running an Analysis for the first time. Note: This option should not be used with Data Stores that have retention processing turned on.</td>
</tr>
<tr>
<td>New Data Since Last Load</td>
<td>Analysis will only read data that is new, since the last time the Data Store Input loaded. If no new data is found, no changes will be made to the Analysis' outputs.</td>
<td>No new data transformations are applied to input data between Analysis Runs, but there is new data to apply transformations to.</td>
</tr>
<tr>
<td>Based on Date Parameters</td>
<td>Analysis will only read data from a specified date range when the Analysis executes.</td>
<td>This setting can allow you to work with a subset of your data, which can be altered by input or Analysis Run to create outputs parsed by date.</td>
</tr>
<tr>
<td>Based on Work ID Parameter</td>
<td>Analysis will only read data from a specified Work ID.</td>
<td>This setting can allow you to work with a subset of your data, which can be altered by input or Analysis Run to create outputs parsed by Work ID.</td>
</tr>
<tr>
<td>Based on File Path Pattern Parameter</td>
<td>Analysis will only read data from files with a file path that matches the pattern parameter.</td>
<td>This setting allows you to work with a subset of your data, sourced only from a file path that matches the pattern used in a parameter.</td>
</tr>
</tbody>
</table>

**Preventing an Analysis from reading data that is past retention period**

When a Data Store is configured with a load range of All, New Since Last Load with no previous execution, New Since Last Load on a rebuild, or By Date with start time of null, the system will check any data being read to make sure that it is within the Data Store's retention period at the time the Analysis starts execution.
Data Retention Process Delay
The system retention process deletes data that is past the retention period only after at least 24 hours have passed since the data was eligible for deletion. This allows any processes that are still reading that data to complete execution.

Parameterized Data Loads
As mentioned above, Date based or Work Id based parameters may be used to control how an Analysis loads its data. As shown in the following examples, these parameters must be created in both the Data Store Input of the Analysis and in the node preceding the Execute Stage Task that will execute the Analysis in the Process Model.

Using Date Parameters
When the Based on Date Parameters Data Load Range Option is used, both a From Date Parameter and To Date Parameter need to be specified. Additionally, these same parameter names need to be used in a Process Model to actually enforce the date range.
Suppose, as illustrated in the top half of the diagram above, that you had an Analysis with a Data Store Input. Imagine that this Data Store Input contained years’ worth of Data, but that you only wanted to pull Data that was loaded into it during a specific date range.

To do so, you would first need to create Date Parameters within the Data Store Input of the Analysis; in the illustration, this is the Analysis at Point b. After that, you’d need to use those parameters in a Process Model.

Within the Process Model, you could create the parameters as User Variables in the node preceding the Execute Stage task that runs the Analysis. In the bottom half of the diagram, this is represented by Point a. These User Variables would need to have the same names as the parameters created in the Analysis at Point b - $\text{toDate}$ and $\text{fromDate}$ - and they could be Literal variables of the Datetime type that equaled literal Dates.

For example, you could set $\text{fromDate}$ to 10/24/2016 and $\text{toDate}$ to 11/24/2016.
With such a configuration, the Execute Stage task at Point c would cause the Analysis at Point c to only load data that was loaded into the Analysis’ Data Store Input between 10/24/2016 and 11/24/2016.

Using Work ID Parameters
Like Date Parameters, Work ID Parameters need to be combined with usage in a Process Model to be properly applied.

Work ID Parameter Example

In the diagram above, we have a configuration where two Analyses output to the same Data Store (c). Suppose we have a third Analysis that uses that Data Store (c) as an input, but in this Analysis we only want to load Data that was pushed to Data Store (c) by Analysis b. To accomplish this, we can use a Work Id Parameter. To do so, you could set Analysis d’s Data Store Input node (c) to have a Data Load Range that is Based on Work ID and create a Work ID Parameter. Here we will call the Work ID Parameter parameterName.
After creating the Work ID Parameter in Analysis d, you would then need to create a Process Model (lower half of diagram above). The Process Model would need to include a Start Stage Event that triggers the Process Model when Data Store a finishes loading (a). It would then need an Execute Stage Task that executes Analysis b.

After the Execute Stage Task, you could then create an Exclusive Gateway. Within the Exclusive Gateway, you would need to create a property reference variable with a name matching the Work ID Parameter you created in Analysis d and a value equal to the work id of the preceding Analysis, Analysis b.

That is, `parameterName = {result}.workId`

This variable would then control how Analysis d loads in the Execute Stage Task for Analysis d. With this setup, Analysis d would only pull data associated with the latest work id of Analysis b, even if Analysis b1 may have also pushed data to Data Store c.
Using a File Path Pattern Parameter Name

In the diagram above, we have a configuration where a Data Store points to an external file system. When using this Data Store as a Data Store Input node in an Analysis, a) you can select a Data Load Range of Based on File Path Pattern Parameter, to only pull in files with a file path that matches a specific pattern. After creating the Analysis, you will need to create a Process Model. Within the Process Model, you will then need to b) create a variable with the same name as the parameter created in the Analysis, within a node that serves as an input to c) an Execute Stage Task that runs the Analysis. When the variable is created in part b, you can set the file path pattern that you want matched as the variable’s value.

After creating such a Process Model, the Analysis referenced by the Execute Stage Task will only pull files into the Data Store Input node from part a that match the file path pattern set in part b - when the Process Model is used to execute the Analysis.
Filter Records Tab

Supported for Data Stores using the following file types: JSON, ORC, Parquet.

The Filter Records tab allows you to create a filter expression for your JSON, ORC, or Parquet file based Data Store at the Data Store Input node level. Filter expressions can be created using the general expression language used throughout Infogix Data3Sixty or SQL.

As an example, consider the following data set.

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Justin</td>
<td>19</td>
</tr>
<tr>
<td>Michael</td>
<td>5</td>
</tr>
<tr>
<td>Andy</td>
<td>30</td>
</tr>
</tbody>
</table>

To return the single record belonging to Andy, you could write the following expression, using either the product’s general expression language or SQL:

\[ \text{age > 19} \]

Doing so would produce the following data set:

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>30</td>
</tr>
</tbody>
</table>

Filter Fields Tab

The Filter Fields tab allows you to filter a Data Store Input of the JSON, ORC, or Parquet format at the field level. If the tab’s functionality is turned on by checking its check box, then only fields that are added to the parameter are those that will be passed through the filter. If the tab’s functionality is turned off, all fields will be passed through the filter.

As an example, consider the following data set.

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Justin</td>
<td>19</td>
</tr>
<tr>
<td>Michael</td>
<td>5</td>
</tr>
<tr>
<td>Andy</td>
<td>30</td>
</tr>
</tbody>
</table>
Turning on the Filter Fields functionality and selecting the *age* field would produce the following result.

```
age
19
5
30
```

*Overflow Field Name*

For data files that contain more fields than are present in the Data Store definition, when lenient parsing is set, an Overflow Field name parameter will become available in the Data Store Input node’s properties panel. Content from the extra fields will then be pushed into this Overflow Field in the JSON format when the Analysis runs.

*Data Store Output*

Data Store Output nodes are Analysis end points. They are used to create or choose a data store that holds the fields manipulated by the Analysis Designer.

Note: This node does not actually execute on a sample of the Data Set when building an Analysis in the UI.

*Selecting a Data Store*

Before designing an Analysis, you can choose to create a data store ahead of time to serve as your output data store. If you have done so, use the data store drop down to select it from the Pipeline where it was saved.

*Creating a New Data Store*

If you haven’t created a Data Store to serve as your output store ahead of time, you can create a new data store from inside the Analysis Designer.

To do so, use the “Create New Data Store” check box, and you can create the store within a Pipeline on the fly.

*Maximum Output Partitions Setting*

This option sets the maximum amount of files that the newly created Data Store Output can be comprised of. These are the files that contain the Data Store’s data, which are available within the Data Store’s View Content Screen.

For performance, Maximum Output Partitions is set to 512 by default; however, you can set it to any number that you’d like. You should note, however, that using a small number of output partitions on a large dataset can cause your Analysis to have a much longer runtime.
Decrypt Secure Values: Admins Only
If you are a system admin and you have secure, encrypted fields within an Analysis, you can opt to decrypt them in your Data Store Outputs by checking the Decrypt Secure Values check box. This option is available for all External Store Types.
Note that encrypted values will not be immediately decrypted in the Data Store Output node’s sheet.

Field Mapping
As data passes from Data Store Inputs, through Enhance and Analytics nodes, and into a Data Store Output, the selected fields may change in structure and in name many times.
Field Mapping allows you to map fields from your final Enhance and/or Analytics nodes to your Data Store Output node.
Fields may be mapped from an Enhance/Analytics node to preexisting fields in a Data Store Output node; or, fields can be newly created in the Output node on the fly.

Update Tab
If you would like to update records within a Data Store based on a unique set of key fields that identify each record, you can do so by checking the Update Data in Data Store check box available in the Field Mapping Tab.
Whenever a Data Store Output node with Update configured receives new data, it will check to see whether the values in a record’s Key Fields already exist in a record within the Data Store. If they do, the Data Store will update all other fields for that record, rather than create a new record.
For example, suppose Update were configured with id set as the Key Field, and the following record existed in the Data Store:

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>123</td>
</tr>
</tbody>
</table>

Since Update is configured, the next time this Data Store loads, it will check to see whether newly loaded records have an id = 001.
Suppose there are records that do:

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>456</td>
</tr>
<tr>
<td>001</td>
<td>789</td>
</tr>
</tbody>
</table>
Rather than create more records for id 001, the Data Store will update the pre-existing record using the last new record with a matching Key Field value that it finds. This will assign the pre-existing record where id = 001 a value of 789.

- **Insert when Record to update doesn’t exist**
  If Update Data in Data Store has been turned on, you may also enable record insertion when a record to update does not exist, by checking Insert when Record to update doesn’t exist. Turning this option on will mean that if the Data Store is set to update, and it encounters a record with new key field values that are not present in the Data Store, then the new record will be inserted into the Data Store.

**Options Tab**
When outputting to an External Data Store of the S3/HDFS/file system layout type, the Options tab allows you to control how files are partitioned in the external file system and whether the files are compressed.

- **Output Writer**
  This parameter determines whether partitioning control is turned on or off. The Enhanced option is what supports partitioning control. The Basic option does not support partitioning control; it uses a deprecated form of partitioning that Analyses used before the Options tab was added.

- **Partition By**
  This parameter allows you to select a set of fields to partition the Data Store’s contents by. Importantly, the field(s) that you select to partition by will not be included in the data file contents; rather, a folder will be created for each unique combination of values within the partitioning field set.
  As a simple example, consider the following data set.

<table>
<thead>
<tr>
<th>name</th>
<th>amount</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>bill</td>
<td>100</td>
<td>A</td>
</tr>
<tr>
<td>ann</td>
<td>200</td>
<td>A</td>
</tr>
<tr>
<td>carl</td>
<td>300</td>
<td>A</td>
</tr>
<tr>
<td>leslie</td>
<td>400</td>
<td>B</td>
</tr>
<tr>
<td>tom</td>
<td>500</td>
<td>B</td>
</tr>
<tr>
<td>stacey</td>
<td>600</td>
<td>B</td>
</tr>
<tr>
<td>ben</td>
<td>700</td>
<td>C</td>
</tr>
<tr>
<td>jill</td>
<td>800</td>
<td>C</td>
</tr>
<tr>
<td>tony</td>
<td>900</td>
<td>C</td>
</tr>
<tr>
<td>quinn</td>
<td>1000</td>
<td>D</td>
</tr>
</tbody>
</table>
Were you to select grade as the Partition By field, you would end up with four folders, A, B, C, and D; and, each folder would contain the records that had the folder’s grade, with only the name and amount fields present. For example, the A folder would look something like this:

![Diagram](image)

**Compression**

This parameter is simply used to control whether the files output to the data store are compressed; and, if they are compressed, what type of compression should be used.

**Case Store Input**

The Case Store Input node allows you to bring records from a Case Store into an Analysis.

The parameters available in the Case Store Input Properties Panel are nearly identical to those described above for a Data Store Input. The only difference is that you will be selecting a Case Store To Read, not a Data Store.

**Case Store Output**

Case Store Output nodes are Analysis end points. Bringing a Case Store Output node into an Analysis is how you can populate a Case Store with header records.

Note: This node does not actually execute on a sample of the Data Set when building an Analysis in the UI.

**Display Name**

This parameter simply sets the node’s display name in the diagram.

**Case Store**

Here, a drop down will allow you to select a predefined Case Store into which to push data. Note that unlike a Data Store Output, there is no option to create a Case Store Output on the fly. In order to use a Case Store Output node, you must have already created the Case Store, and given it the appropriate field structure to which you can map.
Field Mapping Tab
Fields from your Analysis must be mapped to appropriate fields in your Case Store Output. Mapping requires that Incoming Fields have the same data types as Case Store Fields.

Update Tab
If you would like to update records within a Case Store based on a unique set of key fields that identify each record, you can do so by checking the Update Data in Case Store check box available in the Field Mapping Tab.

Whenever a Case Store Output node with Update configured receives new data, it will check to see whether the values in a record’s Key Fields already exist in a record within the Case Store. If they do, the Case Store will update all other fields for that record, rather than create a new record.

For example, suppose Update were configured with id set as the Key Field, and the following record existed in the Case Store:

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>123</td>
</tr>
</tbody>
</table>

Since Update is configured, the next time this Case Store loads, it will check to see whether newly loaded records have an id = 001.

Suppose there are records that do:

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>456</td>
</tr>
<tr>
<td>001</td>
<td>789</td>
</tr>
</tbody>
</table>

Rather than create more records for id 001, the Case Store will update the pre-existing record using the last new record with a matching Key Field value that it finds. This will assign the pre-existing record where id = 001 a value of 789.

Custom Completion Status
The Custom Completion Status node allows you to define a custom status to signify the completion of an Analysis. The intent of this status is to be something that represents more than just a binary success/failure of the Analysis - which is already represented by the Analysis’s result code. When using a Custom Completion Status node, you can define logic within your Analysis to output a status that represents a specific condition. You can then use that status within a Process Model.
When a Custom Completion Status node is added as the endpoint of an Analysis, it will make the values from the first row of every field in the node accessible within a Process Model.

Note: This node does not actually execute on a sample of the Data Set when building an Analysis in the UI.

Creating a Custom Completion Status
For example, suppose you have created an Analysis that concludes by creating a single row representing a number of conditions that are derived from computations performed on fields within previous nodes.

<table>
<thead>
<tr>
<th>condition1</th>
<th>condition2</th>
<th>measure1</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>100</td>
</tr>
</tbody>
</table>

To utilize a Custom Completion Status node, you could create additional columns using expressions that evaluate to String values, which represent conditions.

For example, consider the following expression for a new field called ‘customStatus’:

```
IF((condition1 = true && condition2 = true && measure1 > 90), 'good', 'bad')
```

For this data set, the expression would evaluate to ‘good’:

<table>
<thead>
<tr>
<th>condition1</th>
<th>condition2</th>
<th>measure1</th>
<th>customStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>true</td>
<td>100</td>
<td>good</td>
</tr>
</tbody>
</table>

Using a Custom Completion Status in a Process Model
Within a Process Model, you could then access the customStatus field and its value by creating a literal variable within any node. The variable needs to have the same name as the field’s name in the Analysis. Once this is setup, the variable will contain the same value as is held in the first row of its associated field in the Custom Completion Status node within the Analysis. From here, the variable can then be used to implement logic within the Process Model.
Accessing status variables directly in Notify Tasks

If you do not want to use your status variable to perform any logic within your Process Model, and you simply need to use it in messaging, you may refer to the variable directly by name within a Notify task’s messaging parameters, using the following syntax:

$\{variableName\}$
Sqoop In

The Sqoop In node can be used to efficiently ingest data from an external, relational database. This feature utilizes Apache Sqoop.

To utilize the node, Arguments and values for those Arguments must be set in the node’s property panel. Fields to accept what is returned by the Sqoop In node must also be configured.

Note: This node does not actually execute on a sample of the Data Set when building an Analysis in the UI.

Creating Arguments

To use the Sqoop In node, you must define a set of Arguments and values for each Argument.

Arguments should be valid Apache Sqoop Arguments, preceded by a ‘--’.

For example, to connect to a database via a JDBC URL, you would create a ‘--connect’ Argument. After creating the ‘--connect’ Argument, you would then create a value for it, using the JDBC URL String. For example, ‘jdbc:mysql://172.17.30.206:3306/test’

Within the UI, this argument and value would be specified and listed as follows:
--connect
jdbc:mysql://172.17.30.216:3306/test

Encrypting Arguments

Sqoop In Arguments and their values may also be encrypted. This option should be used, for example, when creating the value for a --password argument.

Arguments used to Connect to a Database

While a full listing of all commands that can be used within Apache Sqoop is beyond the scope of this guide, the following example can be used as a model for how to use this node to make a basic connection to an external, relational database.

--connect
jdbc:mysql://172.17.30.216:3306/test
--username
userRoot123
--password
mypassword
--as-textfile
--compress
--null-string
""
Using the Analysis Designer

Analysis Designer Nodes

--null-non-string
```

--table
customers
--columns
id, name, purchaseAmount
--split-by
id
```

In this example, the arguments --connect, --username, and --password and their respective values are common Apache Sqoop arguments. The arguments --as-textfile, --compress, --null-string, --null-non-string, --table, --columns, and --split-by and their respective values are Apache Sqoop Import Control Arguments.

A full description of these, and other Apache Sqoop Arguments, can be found [here](#).

Fields

In addition to creating Arguments, you must also create Fields in the Sqoop In node, to accept what is received by ingestion and to hold the values you want to process within your Analysis. These fields should match those specified underneath the --columns argument in the Arguments tab.

Sqoop Out

The Sqoop Out node can be used to efficiently output data to an external, relational database. This feature utilizes Apache Sqoop.

To utilize the node, Arguments and values for those Arguments must be set in the node’s property panel. Fields that define which fields should be output to within the external database must also be configured.

Note: This node does not actually execute on a sample of the Data Set when building an Analysis in the UI.

Creating Arguments

To use the Sqoop Out node, you must define a set of Arguments and values for each Argument.

Arguments should be valid Apache Sqoop Arguments, preceded by a ‘--’.

For example, to connect to a database via a JDBC URL, you would create a ‘--connect’ Argument. After creating the ‘--connect’ Argument, you would then create a value for it, using the JDBC URL String. For example, ‘jdbc:mysql://172.17.30.206:3306/test’

Within the UI, this argument and value would be specified and listed as follows:

--connect
jdbc:mysql://172.17.30.216:3306/test

**Encrypting Arguments**

Sqoop Out Arguments and their values may also be encrypted. This option should be used, for example, when creating the value for a `--password` argument.

**Arguments used to Connect to a Database**

While a full listing of all commands that can be used within Apache Sqoop is beyond the scope of this guide, the following example can be used as a model for how to use this node to make a basic connection to an external, relational database.

```
--connect
jdbc:mysql://172.17.30.216:3306/test

--username
userRoot123

--password
mypassword

--input-null-string
"

--table
customers
```

In this example, the arguments `--connect`, `--username`, and `--password` and their respective values are common Apache Sqoop arguments. The arguments `--input-null-string`, and `--table` and their respective values are Apache Sqoop Export Control Arguments.

A full description of these, and other Apache Sqoop Arguments, can be found [here](#).

**Fields**

In addition to creating Arguments, you must also create Fields in the Sqoop Out node, to define which fields within the specified external database and table should be output to. The definitions of the fields from your Analysis should match those to which you are outputting.

**Decrypt Secure Values: Admins Only**

If you are a system admin and you have secure, encrypted fields within an Analysis, you can opt to decrypt them in your Sqoop Out nodes by checking the Decrypt Secure Values check box.

Note that encrypted values will not be immediately decrypted in the Sqoop Out node’s sheet.
6 ■ Using the Analysis Designer

Analysis Designer Nodes

Output Plugin
The Output Plugin node allows you to use a custom, external tool that you have written in Java, outside of the product, within an Analysis. The Output Plugin node differs from the Plugin node in that it can only receive fields from incoming nodes.

For more help with the Output Plugin node, please contact Infogix Support.

Kinesis
The Kinesis node can be used to process live data streams in real-time. For more help with the Kinesis node, please contact Infogix Support.

Excel Output
The Excel Output node allows you to output your Analysis data to a Microsoft Excel .xlsx file. On the cloud version of the product, this file must be pushed to Amazon S3. On the Enterprise version of the product, this file must be pushed to the system’s HDFS or file path.

Details Tab
■ Output Path
This parameter is used to specify the path to the Amazon S3 bucket or the HDFS/file system path that the Excel file will be pushed to.
Formatting of paths is as follows:
■ s3://<bucket_name>/<path>
■ hdfs://<path>
■ file:/// <path>

■ Mapping
The Mapping grid allows you to specify a number of parameters, including:
■ Source
This is a required parameter that is used to specify an input source from which to pull fields of data. Note that the Excel Output node may accept multiple input sources.

■ Array Field
This is an optional parameter that is used to reference an array field within a Source, that you want to use to select specific fields from the Source. The contents of the array field should be representative of the data you want to pull from the Source and output to the Excel file. Note that when using the Array Field parameter, fields will be pushed to the Excel file in the order in which they appear in the array. Without the Array Field parameter, field order is not guaranteed.
- **Default Sheet Name**
  This is a required parameter. The value entered is used as the default value for sheet names in the Excel file when sheet names are not specified with the Sheet Name Field parameter.

- **Sheet Name Field**
  This is an optional parameter that is used to create sheet names in the Excel file. When a Sheet Name Field is specified, the node will create 1 sheet for each unique value in the Sheet Name field, containing the records that correspond to that unique value.

**Basic Excel Output Node Example**
Input Data, aka ‘myData’:

<table>
<thead>
<tr>
<th>field1</th>
<th>field2</th>
</tr>
</thead>
<tbody>
<tr>
<td>bill</td>
<td>c</td>
</tr>
<tr>
<td>jane</td>
<td>b</td>
</tr>
<tr>
<td>bob</td>
<td>a</td>
</tr>
</tbody>
</table>

Output Path: s3://someBucket/somePath
Source: myData
Default Sheet Name: sheet1
Excel Output Data in s3://someBucket/somePath:
Sheet name = sheet1

<table>
<thead>
<tr>
<th>field2</th>
<th>field1</th>
</tr>
</thead>
<tbody>
<tr>
<td>bill</td>
<td>c</td>
</tr>
<tr>
<td>jane</td>
<td>b</td>
</tr>
<tr>
<td>bob</td>
<td>a</td>
</tr>
</tbody>
</table>

**Excel Output Node Example using Array Field**
As shown in the basic example above, without an Array Field specified, outputted field order is not guaranteed to match what is shown in the Excel Output node UI.
To guarantee outputted field order, you could utilize an Array Field.
Suppose for example, we reconstructed the input data, myData, from the example above into the following Array Field.

\[
\text{field3} =
\begin{align*}
&\{\text{fieldNames: "field1", "field2"}, \text{fieldValues: "c", "bill"}\}, \\
&\{\text{fieldNames: "field1", "field2"}, \text{fieldValues: "b", "jane"}\}, \\
&\{\text{fieldNames: "field1", "field2"}, \text{fieldValues: "a", "bob"}\}
\end{align*}
\]

Were we to specify field3 as our Array Field and use the same Source, Output Path, and Default Sheet Name as the example above, we would produce the following Excel Output, with field order matching the field order of our Array Field.

Excel Output Data s3://someBucket/somePath:
Sheet name = sheet1

<table>
<thead>
<tr>
<th>field1</th>
<th>field2</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>bill</td>
</tr>
<tr>
<td>b</td>
<td>jane</td>
</tr>
<tr>
<td>a</td>
<td>bob</td>
</tr>
</tbody>
</table>

**Excel Output Node Example Using Sheet Name Field**

The Sheet Name Field parameter can be used to create sheet names in the Excel file. When a Sheet Name Field is specified, the node will create 1 sheet for each unique value in the Sheet Name field, containing the records that correspond to that unique value. Note that the values in the Sheet Name field will not be included as record values in the Excel output data.

For example, suppose you had the following data set:

<table>
<thead>
<tr>
<th>field0</th>
<th>field1</th>
<th>field2</th>
</tr>
</thead>
<tbody>
<tr>
<td>sheet1</td>
<td>c</td>
<td>bill</td>
</tr>
<tr>
<td>sheet1</td>
<td>b</td>
<td>jane</td>
</tr>
<tr>
<td>sheet2</td>
<td>a</td>
<td>bob</td>
</tr>
</tbody>
</table>

Were you to specify field0 as the Sheet Name Field, you would end up with the following Excel Output:
Sheet name = sheet1

<table>
<thead>
<tr>
<th>field1</th>
<th>field2</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>bill</td>
</tr>
<tr>
<td>b</td>
<td>jane</td>
</tr>
</tbody>
</table>

Sheet name = sheet2

<table>
<thead>
<tr>
<th>field1</th>
<th>field2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>bob</td>
</tr>
</tbody>
</table>

**Options Tab**
The Options tab allows you to further control how files are output to the file system that the Excel Output node pushes data to.

- **Use traditional folder structure when used for output check box**
  This check box allows you to output files to a traditional folder structure as opposed to a Hadoop folder structure.

- **Partition By**
  This parameter allows you to select a set of fields to partition the Excel Output node’s contents by. Importantly, the field(s) that you select to partition by will not be included in the data file contents; rather, a folder will be created for each unique combination of values within the partitioning field set.
  As a simple example, consider the following data set.

<table>
<thead>
<tr>
<th>name</th>
<th>amount</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>bill</td>
<td>100</td>
<td>A</td>
</tr>
<tr>
<td>ann</td>
<td>200</td>
<td>A</td>
</tr>
<tr>
<td>carl</td>
<td>300</td>
<td>A</td>
</tr>
<tr>
<td>leslie</td>
<td>400</td>
<td>B</td>
</tr>
<tr>
<td>tom</td>
<td>500</td>
<td>B</td>
</tr>
<tr>
<td>stacey</td>
<td>600</td>
<td>B</td>
</tr>
<tr>
<td>ben</td>
<td>700</td>
<td>C</td>
</tr>
<tr>
<td>jill</td>
<td>800</td>
<td>C</td>
</tr>
<tr>
<td>tony</td>
<td>900</td>
<td>C</td>
</tr>
<tr>
<td>quinn</td>
<td>1000</td>
<td>D</td>
</tr>
</tbody>
</table>
Were you to select grade as the Partition By field, you would end up with four folders, A, B, C, and D; and, each folder would contain the records that had the folder’s grade, with only the name and amount fields present.
For example, the A folder would look something like this:

```
grade=A
part-00000
  bill 100
  ann 200
  carl 300
```

Including other Analyses within an Analysis

Within the Input/Output node palette, there are 4 specific nodes that can be used to include one Analysis within another Analysis. Described below, these nodes are the Placeholder Input, Placeholder Output, Include Analysis Input, and Include Analysis Output nodes.

**Placeholder Input**
The Placeholder Input node should be used in one Analysis to receive input from another Analysis that is using an Include Analysis Output node.

The Nested Output Analysis node outputs data to the Analysis containing the Placeholder Input node. The Placeholder Input node will receive the output of the Analysis containing the Nested Output Analysis node.
**Placeholder Input Fields**
The fields created within the Placeholder Input’s properties panel should match the data types of the fields being output to the Placeholder Input node by the Include Analysis Output node. You can either create these fields from scratch, or use the Import button to import the field definitions from another, pre-existing data store.

**Placeholder Output**
The Placeholder Output node should be used in one Analysis to output to another Analysis that is using an Include Analysis Input node.

**Placeholder Output Fields**
The fields created within the Placeholder Output’s properties panel will be gathered by another Analysis when using an Include Analysis Input node and selecting the Analysis to Read parameter.

**Nested Input Analysis**
The Nested Input Analysis node receives output from another Analysis that is using a Placeholder Output node. The connection to the Analysis that the Nested Input Analysis node receives data from is made by setting the Analysis to Read parameter within the Nested Input Analysis node’s properties panel.
Nested Input Analysis Example

Utilizing Multiple Outputs from a Nested Input Analysis in Nodes that take multiple Inputs
If you are using a Nested Input Analysis that receives fields from multiple Placeholder Output nodes, the Nested Input Analysis node will create an outgoing port for each Placeholder Output.

If you then attempt to connect these outgoing ports to a node that accepts multiple inputs, you will find that only one of the fields is carried over.

For example, this issue could occur when attempting to connect all outgoing ports from a Nested Analysis Input node (that contains multiple outgoing ports) to a Nested Analysis Output node.

The workaround to this issue is to gather fields from outgoing ports with Select nodes.
For example:
Using Select nodes, you can collect the fields that the Nested Analysis Input node receives from each of its Placeholder Outputs nodes. You can then use the Select nodes as inputs to a common node, where all of the fields can be used.

**Nested Output Analysis**

The Nested Output Analysis node pushes to another Analysis that is using a Placeholder Input node. The connection to the Analysis that the Nested Output Analysis node pushes data to is made by setting the Analysis to Read parameter within the Nested Output Analysis node’s properties panel.

**Placeholder Mappings**

When using the Nested Output Analysis node, you must also map the Nested Output Analysis node’s inputs to Placeholder Inputs from the Analysis to Read.

**Nested Output Analysis Example**

Executing this Analysis will ‘include’ the other Analysis. As a result, this Analysis will push data to the other Analysis.
Enhance Nodes

Sample
The Sample node allows you to define Sampling at any point in an Analysis. The specifications made in the Sample node will apply to the sheet displayed for the node itself and extend to any subsequent nodes in the Analysis’ flow. Note that the Sample node will interact with any Sampling settings made to the Analysis as a whole and at the Data Store Input level. Sampling in the Sample node is always applied after these other settings have taken effect.

For example, with the following combination of settings, a Sample node would display 100 records:

1. Analysis level settings: Max number of records to read: 5000, Sampling Rate (%): 5
   = 250 records
2. Data Store Input settings: Max number of records to read: 250, Sampling Rate (%): 50
   = 125 records
3. Sample node settings: Max number of records to read: 125, Sampling Rate (%): 80
   = 100 records

The sample node can be particularly useful when part of an Analysis contains resource intensive computations and you want to quickly test these in a sheet.

Sample Node During Execution
Sample nodes can also be used to trim down the amount of data that passes through an Analysis when it is executed. This can be particularly useful if an input contains a large amount of data and you’d like to speed up execution time.

Select
The Select node allows you to select Fields from one node, in order to pass these fields onto another node. This can be useful when one node, such as a Data Store Input, has a large amount of fields available but you only want to Enhance or Analyze a few.

Many Analysis Designer nodes have Select functionality, however isolating field Selection into a discrete step can help you stay organized when designing your Analysis. It can also help others who didn’t create the Analysis achieve a better understanding of what is going on.
Using the Analysis Designer  6

Analysis Designer Nodes

Sort
The Sort node sorts fields’ values in ascending or descending order.
Sorting records before placing them into a Data Store Output node can help keep data organized as it is used in other Data Stages.

Rename
The Rename node is used to rename Data Store fields.
After performing transformations, it may be necessary to give fields new names that more accurately categorize the values the fields contain.

Filter
The Filter node is used to apply filter expressions to data store fields. After a filter is applied, a new field is created containing only values that meet the conditions specified in the filter expression.

In
The In node is somewhat of a hybrid between a Join and a Filter node, in that it allows you to combine data from two inputs based on matching records in a chosen field; however, the In node will only return accompanying fields from one of those inputs. This is best illustrated by way of example.
For instance, suppose you had the following two datasets:

**Input 1**

<table>
<thead>
<tr>
<th>id</th>
<th>measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>100</td>
</tr>
<tr>
<td>02</td>
<td>200</td>
</tr>
<tr>
<td>03</td>
<td>300</td>
</tr>
<tr>
<td>04</td>
<td>400</td>
</tr>
<tr>
<td>05</td>
<td>500</td>
</tr>
</tbody>
</table>

**Input 2**

<table>
<thead>
<tr>
<th>id</th>
<th>measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1000</td>
</tr>
<tr>
<td>03</td>
<td>3000</td>
</tr>
<tr>
<td>05</td>
<td>5000</td>
</tr>
</tbody>
</table>
There are two potential ways these datasets can be fed into an In node, and, thus, two potential outcomes. In either case, if $id$ is specified as the Outer Source Field, then only matching records from the $id$ field will be returned. Which records are returned within the $measure$ field, however, will depend upon which input is specified as the “Outer Source.”

Note: When using the In node, the data types of the Inner source and Outer source field need to be the same.
Not In

The Not In node is the In node’s opposite, as it will only return records that are in one data set but not in another. This is again best illustrated by way of example.

For instance, suppose you had the following two datasets:

**Input 1**

<table>
<thead>
<tr>
<th>id</th>
<th>measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>100</td>
</tr>
<tr>
<td>02</td>
<td>200</td>
</tr>
<tr>
<td>03</td>
<td>300</td>
</tr>
<tr>
<td>04</td>
<td>400</td>
</tr>
<tr>
<td>05</td>
<td>500</td>
</tr>
</tbody>
</table>

**Input 2**

<table>
<thead>
<tr>
<th>id</th>
<th>measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>1000</td>
</tr>
<tr>
<td>03</td>
<td>3000</td>
</tr>
<tr>
<td>05</td>
<td>5000</td>
</tr>
<tr>
<td>07</td>
<td>7000</td>
</tr>
<tr>
<td>09</td>
<td>9000</td>
</tr>
</tbody>
</table>

There are two potential ways these datasets can be fed into a Not In node, and, thus, two potential outcomes. In either case, if id is specified as the Outer Source Field, only non-matching records from the id field will be returned. Which non-matching records are returned will in turn depend upon which input is specified as the “Outer Source.”
Using the Analysis Designer

Analysis Designer Nodes

Note: When using the Not In node, the data types of the Inner source and Outer source field need to be the same.

Input 1 specified as Outer Source

Input 2 specified as Outer Source

Records 02 and 04 are not in Input 2, so they are returned by the Not In node.

Records 07 and 09 are not in Input 1, so they are returned by the Not In node.
Plugin

The Plugin node allows you to use a custom, external tool that you have written in Java, outside of the product, within an Analysis. The Plugin node differs from the Output Plugin node in that it can receive fields from incoming nodes and pass fields along to other nodes within an Analysis.

For more help with the Plugin node, please contact Infogix Support.

Compute Node

The Compute Node uses an expression to create new computed fields or change the values of existing fields. Once new fields are added or the values of existing fields are changed, the Compute node’s sheet will display all fields from its input node with the changes you have made.

Creating a new computed field

Using the Compute Node to create a new computed field is effectively the same as adding a computed field in any other node’s sheet. To do so, you simply use the Add button within the Compute node’s properties panel and then give the field a unique Output Field Name that does not match the name of any other pre-existing fields in the Analysis. You then select a data type for the field and create an expression that is used to populate the field’s values.

Changes the values of an existing field.

To use the Compute node to change the values of fields that already exist and have values in the node being used as an input to the Compute node, you simply use the Add button and select an Output Field Name from the list of pre-existing fields. You then create an expression that is used to populate the field’s values.

Nested Analysis

The Nested Analysis node allows you to nest one analysis within another analysis. Depending on the analysis that you are nesting, it can accept one or multiple inputs and produce one or multiple outputs. The connection to the nested analysis is made by setting the Analysis to Read parameter within the Nested Analysis node’s properties panel.

Placeholder Mappings

When using the Nested Analysis node, you must also map the Nested Analysis node’s inputs to Placeholder Inputs from the Analysis to Read.
Combining Nodes

Join
The Join node allows you to combine fields from two separate data stores to create a new sheet made of rows where related fields are matched.

To perform a successful Join, the two chosen fields must be related by a common identity field.

The Analysis Designer features four join types:

* **Inner**
  Only creates a row if the left field value has a matching field value on the right.
**Left Outer**
Creates rows for every left field value and joins any matching right field values. Does not include right field values without a matching left field value.

**Right Outer**
Creates rows for every right field value and joins any matching left field values. Does not include left field values without a matching right field value.

**Full Outer**
Creates rows for all left and right field values, regardless of whether there is a match between left and right.

**Union**
The Union node is used to combine data from two or more input nodes. The combined data sets should have the same structure, although they do not necessarily need to have the exact same field names, due to the Union node’s Map Fields feature.

For example, if you had three sheets, and each sheet contained two columns:

**Input 1**

<table>
<thead>
<tr>
<th>city</th>
<th>measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicago</td>
<td>5000</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>5500</td>
</tr>
<tr>
<td>New York</td>
<td>4750</td>
</tr>
</tbody>
</table>

**Input 2**

<table>
<thead>
<tr>
<th>nameOfCity</th>
<th>cityMeasurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atlanta</td>
<td>3000</td>
</tr>
<tr>
<td>Portland</td>
<td>5125</td>
</tr>
<tr>
<td>Phoenix</td>
<td>4000</td>
</tr>
</tbody>
</table>

**Input 3**

<table>
<thead>
<tr>
<th>cityName</th>
<th>meas.</th>
</tr>
</thead>
<tbody>
<tr>
<td>St. Louis</td>
<td>8000</td>
</tr>
</tbody>
</table>
You could use the Union node to map the `city`, `nameOfCity`, and `cityName` fields to each other, and the `measurement`, `cityMeasurement`, and `meas.` fields to each other to create a new dataset combining the data from each input node.

<table>
<thead>
<tr>
<th>city</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicago</td>
<td>5000</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>5500</td>
</tr>
<tr>
<td>New York</td>
<td>4750</td>
</tr>
<tr>
<td>Atlanta</td>
<td>3000</td>
</tr>
<tr>
<td>Portland</td>
<td>5125</td>
</tr>
<tr>
<td>Phoenix</td>
<td>4000</td>
</tr>
<tr>
<td>St. Louis</td>
<td>8000</td>
</tr>
</tbody>
</table>

*Note that the Map Fields feature will give your Output fields the same names as your leftmost input field by default; however, these names can be changed to anything you want by double-clicking on the Output Fields and typing.

**Co-Group**

The Co-Group node is used to group two inputs based on a common identity field and then join associated fields from each input.

For example, consider the following two sheets, Sheet 1 and Sheet 2:

**Sheet 1**

<table>
<thead>
<tr>
<th>Name</th>
<th>Amount A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>1</td>
</tr>
<tr>
<td>Bob</td>
<td>2</td>
</tr>
<tr>
<td>Alice</td>
<td>3</td>
</tr>
<tr>
<td>Carol</td>
<td>4</td>
</tr>
</tbody>
</table>
Performing Co-Group with the Name field from Sheet 1 as the left input and the Name field from Sheet 2 as the right input would result in the following:

Sheet 3

<table>
<thead>
<tr>
<th>Name</th>
<th>Left</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>{A = 1}, {A = 2}</td>
<td>{B = 5}</td>
</tr>
<tr>
<td>Alice</td>
<td>{A = 3}</td>
<td>{B = 6}, {B = 7}</td>
</tr>
<tr>
<td>Carol</td>
<td>{A = 4}</td>
<td>{B = 8}</td>
</tr>
</tbody>
</table>

Co-Group creates records associating all values for each identity field. Values are grouped as arrays, with a field for each input source. Arrays can then be used by other nodes, in computations.

**SQL**

The SQL node can be used to apply SQL based statements to your data set. To use this node, you’ll first need to specify a Schema for each node that enters the SQL node. This will allow you to treat incoming nodes like tables, to which you can apply SQL statements. Once you have created Schemas, you can then specify which fields you’d like to work with by adding *Input fields* within the SQL Properties Panel Outputs tab.
If you plan to create new fields using SQL, you’ll also need to use the Add Field dropdown to create a New Field for each field you want to create.

**SQL Script Editing**

Once you have created Schemas, Input Fields, and New Fields, you will have an outline of the components you can work with within your SQL script.

To create a new Script, use the Edit Script button. Doing so will produce a blank editor into which you can type your SQL Queries.

Keep in mind:
- You will need to use the Schema names that you’ve defined to refer to different input sources as if they were tables. For example, SELECT * FROM schema_A.
- You can type **CTRL+SPACE** for a full list of SQL keywords and functions. (Typing **CTRL+SPACE** after you’ve already typed letters can also function as auto-complete).
- Avoid putting a semicolon (;) at the end of statements - it is not required.

**Execute Query in DB**

The Execute Query in DB node allows you to take the contents of a data set that you have placed in a Data Store and use those contents to query an external database. Results from the query are then returned to the node, where they can be further used in your Analysis.
Details tab

The Details tab is used to construct your SQL query and to specify how (and what) incoming data should be loaded.

Edit SQL

The Edit SQL button allows you to construct a query that will be executed in the external database that the node is configured to point to. This query will be made when the Analysis executes.

When using field names from incoming nodes, the following syntax should be used:

${fieldName}

Note: When referring to string fields in SELECT statements, single quotes are needed. For example: ‘${fieldName}

As an example of using the Edit SQL feature, consider the following data set:

Table Names Data Set

<table>
<thead>
<tr>
<th>TableName</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME_RECON_FILES_CDATA</td>
</tr>
<tr>
<td>ACMECORP</td>
</tr>
<tr>
<td>ACME_PROMO_CDATA</td>
</tr>
</tbody>
</table>

To use the contents from each row in the TableName field within a query, you would refer to the TableName field as ${TableName}. When the query executed, it would then run the query for each row in the dataset, substituting the TableName field value for ${TableName}.

For example, with the data set above, the query:

SELECT * FROM ${TableName}

Would run the following 3 queries during execution.

SELECT * FROM ACME_RECON_FILES_CDATA
SELECT * FROM ACMECORP
SELECT * FROM ACME_PROMO_CDATA

As discussed below, further modification of the query and the creation of Output fields in the Outputs tab would allow you to return the results of the query to the node.
Edit Sample SQL
Like the Edit SQL option, the Edit Sample SQL option allows you to write queries using the contents of fields from incoming nodes. The difference between the two, however, is that Edit Sample SQL only uses content from the Analysis sample (i.e., the values shown in the Analysis’ sheets). Edit SQL, on the other hand, is queried against the entire database and is what is run during the Analysis’ execution.

Effectively, this means: 1) that if you write an Edit Sample SQL script, its results will be shown in the Execute Query in DB node’s sheet during Analysis editing; 2) if no Edit Sample SQL script is written, the results of the Edit SQL script will be shown in the sample sheet instead; 3) and, regardless of whether an Edit Sample SQL script is used, the Edit SQL script will always be used during execution.

Array Field to Process
The Execute Query in DB node can also process up to one incoming array field, and use the contents of that field in its queries. Incoming array fields may come from other Analysis nodes that output them, such as a Co-Group node or a Script node.

Once the array field is selected, it can be used in queries just like other fields, using the same `${fieldName}` syntax.

Data Load Range

All
Uses all data in the table, regardless of any last update time stamp.

New Data Since Last Execution
Only uses data that has been added to the database table since the last time the Analysis ran.

If you would like your query to only pull new data since the last time the analysis executed, you will also need to include one of the following in a WHERE clause:

- WHERE <fieldContainingLastUpdateTime> BETWEEN ${fromTimestamp} AND ${toTimestamp}
- WHERE <fieldContainingLastUpdateTime> >= ${fromTimestamp} and <fieldContainingLastUpdateTime> < ${toTimestamp}

Additionally, <fieldContainingLastUpdateTime> must be in the UTC time zone or converted to the UTC time zone within the query.

In these queries, <fieldContainingLastUpdateTime> refers to a field in your database table that contains last update time stamps. ${fromTimestamp} and ${toTimestamp} refer to system fields that allow the Execute Query in DB node to calculate the Data Load range, based on the setting you choose in the Data Load Range dropdown menu. After adding the WHERE clause to your query, you should make the New Data Since Last Execution selection.
**Isolate each record for processing**
If checked, the Execute Query in DB node will open a database connection for each record that is processed. If unchecked, the node will create one connection per data partition.

**Database**
The Database tab is used to connect to an external database. The database specified here is where the query created in the Details tab will execute.

**Outputs**
The Outputs tab is used to create fields that will hold the results of queries made in the Details tab. Output fields can then be referenced in your SQL query using the AS keyword. For example, consider the following dataset:

**Table Names Data Set**

<table>
<thead>
<tr>
<th>TableName</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACME_RECON_FILES_CDATA</td>
</tr>
<tr>
<td>ACMECORP</td>
</tr>
<tr>
<td>ACME_PROMO_CDATA</td>
</tr>
</tbody>
</table>

To find the row count of each table that is named in this data set, you could first use the Outputs tab to create a New Field called ROWCOUNT.

Within the Details tab, you could then write the following query:

```sql
SELECT COUNT(*) AS ROWCOUNT FROM ${TableName}
```

The results of this query, displayed in the Execute Query in DB node’s sheet, would simply be the number of rows contained in each table from the TableName data set. This could, for example, look something like this:

**Table Names Query Result Set**

<table>
<thead>
<tr>
<th>ROWCOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
</tr>
<tr>
<td>91</td>
</tr>
<tr>
<td>18</td>
</tr>
</tbody>
</table>

**New Field vs. Input Field**
As with other Analysis nodes and features throughout the system, the Outputs tab will let you add either New Fields or Input Fields.
A New Field is a field that will be newly created by the result of the node's query (referred to in the SQL script with an AS operator). You must create a New Field for every field you wish to create with your query.

An Input Field, on the other hand, refers to fields that are entering the Execute Query in DB node from other nodes. If you would like to use one of these incoming fields in your SQL script, you must create an Input field for it.

**Group**

The Group node allows you to group records by one or multiple fields, in order to produce an associated array. The arrays that are associated to each group can then be processed using other Analysis features, such as array functions or the Javascript node.

*Grouped Output Field Name*

This is the name of the new field that will be produced by your grouping and will contain one array for each group.

*Fields to Group By*

This is the set of fields that will be used to create the groups.

**Group Node Example**

Suppose you had the following dataset.

<table>
<thead>
<tr>
<th>purpose</th>
<th>loan_amnt</th>
<th>installment</th>
</tr>
</thead>
<tbody>
<tr>
<td>credit_card</td>
<td>1000</td>
<td>150</td>
</tr>
<tr>
<td>credit_card</td>
<td>1500</td>
<td>200</td>
</tr>
<tr>
<td>credit_card</td>
<td>2000</td>
<td>250</td>
</tr>
<tr>
<td>home_improvement</td>
<td>5000</td>
<td>300</td>
</tr>
<tr>
<td>home_improvement</td>
<td>10000</td>
<td>400</td>
</tr>
<tr>
<td>home_improvement</td>
<td>20000</td>
<td>900</td>
</tr>
<tr>
<td>small_business</td>
<td>10000</td>
<td>500</td>
</tr>
<tr>
<td>small_business</td>
<td>15000</td>
<td>750</td>
</tr>
<tr>
<td>small_business</td>
<td>17500</td>
<td>850</td>
</tr>
</tbody>
</table>
If you were to create a Grouped Output Field named *amounts* and used *purpose* as the Field to Group By, the Group node would produce the following output.

<table>
<thead>
<tr>
<th>purpose</th>
<th>amounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>credit_card</td>
<td>[{loan_amnt: 1000, installment: 150},</td>
</tr>
<tr>
<td></td>
<td>{loan_amnt: 1500, installment: 200},</td>
</tr>
<tr>
<td></td>
<td>{loan_amnt: 2000, installment: 250}]</td>
</tr>
<tr>
<td>home_improvement</td>
<td>[{loan_amnt: 5000, installment: 300},</td>
</tr>
<tr>
<td></td>
<td>{loan_amnt: 10000, installment: 400},</td>
</tr>
<tr>
<td></td>
<td>{loan_amnt: 20000, installment: 900}]</td>
</tr>
<tr>
<td>small_business</td>
<td>[{loan_amnt: 1000, installment: 500},</td>
</tr>
<tr>
<td></td>
<td>{loan_amnt: 15000, installment: 750},</td>
</tr>
<tr>
<td></td>
<td>{loan_amnt: 17500, installment: 850}]</td>
</tr>
</tbody>
</table>

With this newly structured data set, you could process the *amounts* field as an array. For example, you could pass the *amounts* field into a Javascript node and run the following script.

```javascript
var sum = 0;
for(var i = 0; i< input.amounts.length; i++){
    sum += input.amounts[i].loan_amnt;
}
output.total_loan_amnt = sum;
output.purpose = input.purpose;
```

Such a script would produce the following dataset.

<table>
<thead>
<tr>
<th>purpose</th>
<th>total_loan_amnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>credit_card</td>
<td>4500</td>
</tr>
<tr>
<td>home_improvement</td>
<td>35000</td>
</tr>
<tr>
<td>small_business</td>
<td>42500</td>
</tr>
</tbody>
</table>

**Grouping by Multiple Fields**

While the example above demonstrates grouping by a single field, you can also use the Group node to group by multiple fields. For example, consider a scenario where our original dataset had an additional field called *grade*.
Using both *purpose* and *grade* as Fields to Group By would produce the following dataset:

<table>
<thead>
<tr>
<th>purpose</th>
<th>grade</th>
<th>loan_amnt</th>
<th>installment</th>
</tr>
</thead>
<tbody>
<tr>
<td>credit_card</td>
<td>A</td>
<td>1000</td>
<td>150</td>
</tr>
<tr>
<td>credit_card</td>
<td>B</td>
<td>1500</td>
<td>200</td>
</tr>
<tr>
<td>credit_card</td>
<td>B</td>
<td>2000</td>
<td>250</td>
</tr>
<tr>
<td>home_improvement</td>
<td>A</td>
<td>5000</td>
<td>300</td>
</tr>
<tr>
<td>home_improvement</td>
<td>A</td>
<td>10000</td>
<td>400</td>
</tr>
<tr>
<td>home_improvement</td>
<td>C</td>
<td>20000</td>
<td>900</td>
</tr>
<tr>
<td>small_business</td>
<td>B</td>
<td>10000</td>
<td>500</td>
</tr>
<tr>
<td>small_business</td>
<td>B</td>
<td>15000</td>
<td>750</td>
</tr>
<tr>
<td>small_business</td>
<td>D</td>
<td>17500</td>
<td>850</td>
</tr>
</tbody>
</table>

Using both *purpose* and *grade* as Fields to Group By would produce the following dataset:

<table>
<thead>
<tr>
<th>purpose</th>
<th>grade</th>
<th>amounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>credit_card</td>
<td>A</td>
<td>[loan_amnt: 1000, installment: 150]</td>
</tr>
<tr>
<td>credit_card</td>
<td>B</td>
<td>[loan_amnt: 1500, installment: 200],</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[loan_amnt: 2000, installment: 250]]</td>
</tr>
<tr>
<td>home_improvement</td>
<td>A</td>
<td>[loan_amnt: 5000, installment: 300],</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[loan_amnt: 10000, installment: 400]]</td>
</tr>
<tr>
<td>home_improvement</td>
<td>C</td>
<td>[loan_amnt: 20000, installment: 900]]</td>
</tr>
<tr>
<td>small_business</td>
<td>B</td>
<td>[loan_amnt: 10000, installment: 500],</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[loan_amnt: 15000, installment: 750]]</td>
</tr>
<tr>
<td>small_business</td>
<td>D</td>
<td>[loan_amnt: 17500, installment: 850]]</td>
</tr>
</tbody>
</table>

*Group Node Limitations*
Due to browser memory, there is a limit to how many records the Group node can handle per array. You should therefore take care to minimize the amount of records that will be placed in each group.
Shape Nodes

Group By

The Group By node can be used to group rows by a specified field. If there are duplicate values in the specified field, Group By will group those values into a single row.

Group By is useful when working with rows that contain field values you wish to aggregate.

For example, performing Group By on the Name field in the following sheet:

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>$1,000</td>
</tr>
<tr>
<td>Bob</td>
<td>$2,000</td>
</tr>
<tr>
<td>Alice</td>
<td>$3,000</td>
</tr>
</tbody>
</table>

Would Group all duplicate Names into a single record, displaying the two records of Bob and Alice in the Group By node’s sheet:

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
</tr>
<tr>
<td>Alice</td>
</tr>
</tbody>
</table>

At the same time, Group By would Group all Salary values for each record into one array per record.

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>[{Salary = $1,000, Salary = $2,000}]</td>
</tr>
<tr>
<td>Alice</td>
<td>[{Salary = $3,000}]</td>
</tr>
</tbody>
</table>

These array records will not be displayed in the Group By node’s sheet; however, you could add a New Column to the sheet, that uses a function to aggregate these values.

For example, SUM(Salary), which would produce:
6 ■ Using the Analysis Designer

Analysis Designer Nodes

Distinct

The Distinct node is used to remove duplicate records from a specified field. For example, using the Distinct node on the Name field in the following sheet:

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>$3,000</td>
</tr>
<tr>
<td>Alice</td>
<td>$3,000</td>
</tr>
</tbody>
</table>

Would return a sheet containing only one distinct record for “Bob.”

The Distinct node is similar to the Group By node in that it groups duplicate records in a specified field. Unlike Group By, however, Distinct does not allow for subsequent aggregation.

Flatten

The Flatten node is used to return a single, ordered list of all elements within an array.

For example, applying Flatten to:

\[
[[a, b, c, d], [e, f, g, h], [i, j, k, l]]
\]

Would return:

\[
[a, b, c, d, e, f, g, h, i, j, k, l]
\]
Bin
The Bin node allows you to “bin” relatively continuous numeric values into collective output fields to be used instead.
For example, you could apply the Bin node to a Customer Age field to create a new field where each record represents an age range.
Consider a data set where you had to separate children from adults, using age.

<table>
<thead>
<tr>
<th>id</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>12</td>
</tr>
<tr>
<td>002</td>
<td>17</td>
</tr>
<tr>
<td>003</td>
<td>18</td>
</tr>
<tr>
<td>004</td>
<td>22</td>
</tr>
<tr>
<td>005</td>
<td>21</td>
</tr>
</tbody>
</table>

Here you could create one bin from 0 to 18 with an output value of 0 and another bin from 18 to 25 with an output value of 1. This would bin the data in the following manner:

<table>
<thead>
<tr>
<th>id</th>
<th>age</th>
<th>binField</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>002</td>
<td>17</td>
<td>0</td>
</tr>
<tr>
<td>003</td>
<td>18</td>
<td>1</td>
</tr>
<tr>
<td>004</td>
<td>22</td>
<td>1</td>
</tr>
<tr>
<td>005</td>
<td>21</td>
<td>1</td>
</tr>
</tbody>
</table>

*Bin To Values are Not Inclusive*
Note that although the first bin ranges from 0 to 18, the value 18 is placed in the second bin for adults. This is because the To values in bin ranges are not inclusive.

*Missing Value Parameter (optional)*
The Bin node also gives you the option to specify a Missing Value parameter. If specified, this is the value that will be used for any incoming value that does not fall within the range of any bin in your configuration.
Continuing the age example above, 26 would be one such value. Were we to specify 2 as the Missing Value parameter, 26 and all other values that did not fall within the ranges of our bins would be assigned a value of 2 in the generated binField field. On the other hand, if we were to leave Missing Value blank, these incoming values would be assigned a value of null in the binField field.

**Script**

The Script node is used to apply more advanced transformations to inputs from other nodes, using JavaScript.

*Using Auto-Complete*

Typing **CTRL+SPACE** in the JavaScript Editor will activate auto-complete. Depending on what else is typed, auto-complete will display Fields and/or Functions with Help Content.

*Input Fields*

Once an input node has been connected to the Script node’s incoming port, you must use the Add Field drop down to select Input fields. Once Input fields are added, you can use them in your script.

Within your script, Input fields can be referred to using the following syntax:

`input.fieldName`

*Output Fields*

To create a new output field, you must use the Add Field drop down to create a New Field. To use a New Field in your script, use the following syntax:

`output.fieldName`

*Passing Input Fields through the Script node*

When working with the Script node, you need to include all fields that you wish to output from it in your Script.

If there are fields from an incoming node that you simply need to “pass through” without transforming, you will need to select these fields as Input Fields and then use the following statement in your script:

`output.fieldName = input.fieldName;`

This statement should be repeated for each input field you need to pass through. Also note that for fields passed using the statement above, you *do not* need to create Output fields via the Add Field drop down.
**Passing Input Fields with copyInputsToOutputs()**

Another way to pass fields from an input node into a JavaScript node is with the `copyInputsToOutputs()` function available within the Script editor.

**Pass all fields without a RegEx Parameter**

If you simply want to pass all fields from an input node into a JavaScript node, place the following line at the top of your script:

```javascript
copyInputsToOutputs();
```

Using the `copyInputsToOutputs()` function is effectively the equivalent of writing

```javascript
output.fieldName = input.fieldName
```

for every field from your input node.

**Pass specific fields with a RegEx Parameter**

`copyInputsToOutputs()` can also take a regex parameter. Providing such a parameter will tell the function to test whether field names match the regex. Only field names that match the regex will be passed.

For example, if you input fields with the names YearStarted, YearEnded, and MonthRenewed, placing the following in your script would only create outputs for the first two fields:

```javascript
var r = new RegExp('Year\S+');
copyInputsToOutputs(r);
```

For more on RegExp syntax in JavaScript, see here.

**Passing fields with a Join**

Fields may also be “passed” through a Script node through use of a Join node.

In this setup, you would only need to include new output fields and a field to Join on in your script. You could then Join the Script node’s outputs with fields from your original Data Store.
This type of setup could come in useful if you have a large number of fields to pass through a Script node, and you don’t necessarily have the patience to include each one in your script!

*Equality type when comparing variables in Scripts (=== vs. ==)*

When comparing Fields and Variables in the Script node, it is important to consider data and equality types you are using.

For Strict equality, use the `===` operator.

For Lenient equality, use the `==` operator.

*Binding Functions*

In addition to the data transformation functions available throughout Infogix Data3Sixty, the JavaScript node also features some Binding Functions to help you move or change Fields and Records.

- **copyInputsToOutputs()**
  
  As discussed on page 129, this function passes fields from an input node to the JavaScript node.

- **dropRecord()**
  
  This function will remove records from a data set and is best combined with conditional logic. For example, consider the following data set.

<table>
<thead>
<tr>
<th>id</th>
<th>topic</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>machineLearning</td>
<td>98</td>
</tr>
<tr>
<td>2</td>
<td>dataVisualization</td>
<td>96</td>
</tr>
<tr>
<td>3</td>
<td>hangGliding</td>
<td>50</td>
</tr>
</tbody>
</table>

  You could remove the third record from this data set using the following in your script:

  ```javascript
  if(input.topic === 'hangGliding') {
    dropRecord();
  }
  ```

  Within the Script node, the data set would then become:

<table>
<thead>
<tr>
<th>id</th>
<th>topic</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>machineLearning</td>
<td>98</td>
</tr>
<tr>
<td>2</td>
<td>dataVisualization</td>
<td>96</td>
</tr>
</tbody>
</table>

  Keep in mind that without conditional logic, `dropRecord()` will drop every record in the data set.
emitRecord()
This function can be used to change the field values of records. Input parameters are the field names and values to write to them.
For example, consider the following data set once again.

<table>
<thead>
<tr>
<th>id</th>
<th>topic</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>machineLearning</td>
<td>98</td>
</tr>
<tr>
<td>2</td>
<td>dataVisualization</td>
<td>96</td>
</tr>
<tr>
<td>3</td>
<td>hangGliding</td>
<td>50</td>
</tr>
</tbody>
</table>

You could change values in the third record of this data set using the following in your script:

```javascript
if(input.rank<=50) {
    emitRecord(output.topic = 'classification', output.rank = 95);
}
```

Within the Script node, the data set would then become:

<table>
<thead>
<tr>
<th>id</th>
<th>topic</th>
<th>rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>machineLearning</td>
<td>98</td>
</tr>
<tr>
<td>2</td>
<td>dataVisualization</td>
<td>96</td>
</tr>
<tr>
<td>3</td>
<td>classification</td>
<td>95</td>
</tr>
</tbody>
</table>

Keep in mind that without conditional logic, `emitRecord()` will change the specified parameters for every record in the data set.

**Using the Script Input and Script Output tab to handle Javascript’s Numeric Limitations**

When using the Javascript node it is important to be aware of the language’s limitations, particularly when dealing with numeric fields.

- The maximum numeric integer that can be reliably used in Javascript is 9007199254740991. Any number greater than this may potentially be truncated when using the Javascript node.
- To avoid truncation of field values greater than 9007199254740991, you will need to tell the Javascript node to treat a field’s values as Strings. To do so, use the Javascript node’s Script Input and Script Output tabs to mark the field as a String type. If you would still like to treat the field as an Integer or Big Integer outside of the Javascript node, you can mark the field as such in the Javascript node’s Outputs tab.
6 ■ Using the Analysis Designer

Analysis Designer Nodes

Script Rules tab
Here, you can incorporate preexisting Script Rules from Rule Libraries into your Script node. Doing so will allow you to call functions from the incorporated rules by name, within your Script. To incorporate a Script Rule, you just need to identify its parent Rule Library and Rule Group, and then select the Script Rule you want to use.

Auto Number
This useful node allows you to give row numbers to a preexisting dataset that does not have them. Row numbering starts at 0.

Row Number Field Name is simply the name you want to give to the new field containing the row numbers.

Sort Order for Numbering allows you to specify a field to sort by before numbering.

For example: Specifying row as the Row Number Field Name and state (Ascending) as the Sort Order for Numbering Field on the following data set,

<table>
<thead>
<tr>
<th>state</th>
<th>city</th>
<th>measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Illinois</td>
<td>Chicago</td>
<td>98</td>
</tr>
<tr>
<td>Arkansas</td>
<td>Little Rock</td>
<td>96</td>
</tr>
<tr>
<td>New York</td>
<td>Buffalo</td>
<td>95</td>
</tr>
</tbody>
</table>

Would result in the following numbered data set, sorted by state name in ascending order.

<table>
<thead>
<tr>
<th>state</th>
<th>city</th>
<th>measure</th>
<th>row</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arkansas</td>
<td>Little Rock</td>
<td>96</td>
<td>0</td>
</tr>
<tr>
<td>Illinois</td>
<td>Chicago</td>
<td>98</td>
<td>1</td>
</tr>
<tr>
<td>New York</td>
<td>Buffalo</td>
<td>95</td>
<td>2</td>
</tr>
</tbody>
</table>

Tip: Auto Number can be particularly useful if you need to generate a unique id for each record in your data set.

Reshape
The Reshape node allows you to convert rows into columns or columns into rows.
Converting Rows into Columns
At times, you may find it useful to convert a field’s records into fields themselves. This option allows you to do so, by selecting Fields to Group By and Fields to Move to Columns. Once Fields to Group By have been selected, Fields that have been moved to Columns that are part of the same group will appear in the same row.

For example, consider performing a Reshape on the following sample from a data set:

Sample of Loan Data Set Before Reshape

<table>
<thead>
<tr>
<th>id</th>
<th>loan_amnt</th>
<th>term</th>
<th>int_rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>19200</td>
<td>60 months</td>
<td>13.99</td>
</tr>
<tr>
<td>002</td>
<td>7200</td>
<td>36 months</td>
<td>12.29</td>
</tr>
<tr>
<td>003</td>
<td>8000</td>
<td>36 months</td>
<td>14.65</td>
</tr>
<tr>
<td>004</td>
<td>8175</td>
<td>36 months</td>
<td>16.99</td>
</tr>
<tr>
<td>005</td>
<td>8000</td>
<td>36 months</td>
<td>13.99</td>
</tr>
</tbody>
</table>

Selecting term as the Field to Group By, id as the Field to Move to Columns, and 5 as the Number of Rows to Move to Columns would reshape the data set into the following:

Sample of Loan Data Set After Reshape - Grouped By a Single Field

<table>
<thead>
<tr>
<th>loan_amnt</th>
<th>term</th>
<th>int_rate</th>
<th>id</th>
<th>id_2</th>
<th>id_3</th>
<th>id_4</th>
<th>id_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>7200</td>
<td>36 months</td>
<td>12.29</td>
<td>002</td>
<td>003</td>
<td>004</td>
<td>005</td>
<td>009</td>
</tr>
<tr>
<td>19200</td>
<td>60 months</td>
<td>13.99</td>
<td>001</td>
<td>014</td>
<td>101</td>
<td>089</td>
<td>045</td>
</tr>
</tbody>
</table>

In this reshaped data set, the first 5 ids that have a 36 month term have been turned into columns and the first 5 ids that have a 60 month term have been turned into columns. This is highlighted by the values in bold. Additionally, ids that were not listed in the Sample of Loan Data Set Before Reshape were found; for the sake of space, this example assumes that the data set is much larger than what is listed.

It is also important to note that the loan_amnt and int_rate values listed in each record are only applicable to the first id, because we have not grouped by these fields. To obtain records where loan_amnt, term, and int_rate all apply to the ids listed in each record, we would need to Group By multiple fields, which the Reshape node does allow.

Doing so could result in the following data set, where each id matches the loan_amnt, term, and int_rate in its record.
Using the Analysis Designer

Analysis Designer Nodes

Sample of Loan Data Set After Reshape - Grouped By Multiple Fields

<table>
<thead>
<tr>
<th>loan_amnt</th>
<th>term</th>
<th>int_rate</th>
<th>id</th>
<th>id_2</th>
<th>id_3</th>
<th>id_4</th>
<th>id_5</th>
</tr>
</thead>
<tbody>
<tr>
<td>19200</td>
<td>60 months</td>
<td>13.99</td>
<td>001</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7200</td>
<td>36 months</td>
<td>12.29</td>
<td>002</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8000</td>
<td>36 months</td>
<td>14.65</td>
<td>003</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8175</td>
<td>36 months</td>
<td>16.99</td>
<td>004</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8000</td>
<td>36 months</td>
<td>13.99</td>
<td>005</td>
<td>149</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Converting Columns into Rows

At other times, you may wish to convert columns into rows, which the Reshape node also allows.

For example, you may have a data set where records may have either an id or member_id, and you want to merge these columns into a single field comprised of rows from each field.

Example 1: Data Set Before Column to Row Reshape

<table>
<thead>
<tr>
<th>id</th>
<th>member_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td></td>
</tr>
<tr>
<td>009</td>
<td></td>
</tr>
</tbody>
</table>

Specifying id and member_id as Fields to Move to Rows and Name of New Field to Move Columns to as common_id would reshape the data set into the following:
Example 1: Data Set After Column to Row Reshape

<table>
<thead>
<tr>
<th>common_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
</tr>
<tr>
<td>143</td>
</tr>
<tr>
<td>009</td>
</tr>
</tbody>
</table>

Note that record ordering will follow the order in which Fields to Move to Rows are listed in the Properties panel. Also note that blank/null records have been carried over into the common_id column; however, these could easily be removed by using a Filter node.

An example using multiple columns can further illustrate how converting columns to rows will reshape your data set. Consider the following sample.

Example 2: Data Set Before Column to Row Reshape

<table>
<thead>
<tr>
<th>state</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indiana</td>
<td>25</td>
<td>50</td>
<td>75</td>
<td>100</td>
</tr>
<tr>
<td>Maryland</td>
<td>50</td>
<td>100</td>
<td>150</td>
<td>200</td>
</tr>
<tr>
<td>California</td>
<td>500</td>
<td>1000</td>
<td>1500</td>
<td>2000</td>
</tr>
<tr>
<td>Wisconsin</td>
<td>100</td>
<td>200</td>
<td>300</td>
<td>400</td>
</tr>
<tr>
<td>Virginia</td>
<td>50</td>
<td>100</td>
<td>150</td>
<td>200</td>
</tr>
<tr>
<td>Illinois</td>
<td>1000</td>
<td>2000</td>
<td>4000</td>
<td>5000</td>
</tr>
</tbody>
</table>
Specifying Q1, Q2, Q3, and Q4 as Fields to Move to Rows, creating a field named Amount for the Name of New Field to Move Columns to parameter, and creating a field named Quarter for the Name of New Field to Move Column Labels to parameter would result in the following data set.

**Example 2: Sample of Data Set After Column to Row Reshape**

<table>
<thead>
<tr>
<th>state</th>
<th>Amount</th>
<th>Quarter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indiana</td>
<td>25</td>
<td>Q1</td>
</tr>
<tr>
<td>Indiana</td>
<td>50</td>
<td>Q2</td>
</tr>
<tr>
<td>Indiana</td>
<td>75</td>
<td>Q3</td>
</tr>
<tr>
<td>Indiana</td>
<td>100</td>
<td>Q4</td>
</tr>
<tr>
<td>Maryland</td>
<td>50</td>
<td>Q1</td>
</tr>
<tr>
<td>Maryland</td>
<td>100</td>
<td>Q2</td>
</tr>
</tbody>
</table>

Note that the four quarter columns have been consolidated into a single column, and that the labels of these previous four columns are now used as the values for the column’s records. As a result, each state now has four records - one for each quarter. Similarly, the values that filled the quarter columns before the reshape now populate the Amount column, each value corresponding to the quarter and state it was associated with before the reshape.

Also note that for brevity, only six records are shown in the table above. After this reshape, the full data set would actually contain 4 records for each state (one record for each quarter column), for a total of 24 records.

**Split Node**

The Split node allows you to parse a String field, using Regular Expressions. Doing so requires 4 input parameters:

- **Field to Split**: This is the String field you would like to split into multiple fields.
- **Number of Fields**: This is the number of fields you would like to split the Field to Split into, and the number of new fields that will be added to your data set.
- **Field Names**: These are the names of the new fields created by the split. Names default to Field1, Field2, Field3, etc., but can be changed by double clicking.
- **Regular Expression Pattern**: This is the RegExp you want to use to split your field. This is the most crucial component of the Split node, as it is what actually does the work of splitting the field. As such, it is covered in more detail below.
Split node Regular Expression Patterns

The Split node follows the rules of Regular Expressions in Java. This means that splitting is not always simply a matter of specifying the characters with which you want to split.

For example, if you had a name field that you wanted to split into a firstName field and a lastName field, you could split using the Space character - simply by typing the Space bar once - or you could use the more formal RegExp version of Space, \s.

Either case would result in the following:

**Split Node Example 1 - Splitting on a Space**

<table>
<thead>
<tr>
<th>name</th>
<th>firstName</th>
<th>lastName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ernest Hemingway</td>
<td>Ernest</td>
<td>Hemingway</td>
</tr>
</tbody>
</table>

Some cases require more special treatment, however. For example, if you wanted to parse a URL using the period character, and you simply used the period character . as your Regular Expression Pattern, you would very quickly find that this does not work.

**Split Node Example 2 - A RegExp That Doesn’t Work**

<table>
<thead>
<tr>
<th>url</th>
<th>protocolPrefix</th>
<th>companyName</th>
<th>worldWideSuffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>http://</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><a href="http://www.infogix.com">www.infogix.com</a></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This is because with Regular Expressions the period . is a special character, which needs to be escaped. In Infogix Data3Sixty, special RegExp characters should be escaped using a double back slash \\.

Using such a RegExp would result in \\. which would properly split our URL field into the three fields shown below.

**Split Node Example 3 - Splitting on \\**

<table>
<thead>
<tr>
<th>url</th>
<th>protocolPrefix</th>
<th>companyName</th>
<th>worldWideSuffix</th>
</tr>
</thead>
<tbody>
<tr>
<td>http://</td>
<td><a href="http://www">http://www</a></td>
<td>infogix</td>
<td>com</td>
</tr>
<tr>
<td><a href="http://www.infogix.com">www.infogix.com</a></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Infogix Data3Sixty uses the Java Regular Expression Standards. More on this can be found here:

https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html
Note: In both of these examples, the characters that are used to split on are effectively removed from the newly created columns.

**REST API Call Node**

The REST API Call node allows a REST service to be read from or written to, making a REST call for each input record that is fed to it. This node is appropriate only for cases where the data returned/updated by each REST call is a few thousands records or less; it is not appropriate for cases where the data returned/updated by each REST call is in the tens of thousands of records or higher. Additionally, this node does not support file input/output (upload/download) to HTTP calls.

Note that to form a JSON request body or to parse a JSON response, it is recommended that a Script node be used before or after this node. The JSON.parse and JSON.stringify Javascript methods can be used within such a script.

**Request tab**

This tab is used to define your request to the REST service. It contains the following parameters.

- **URL**
  The URL to the REST service. Note that the URL can contain field value references using the ${} syntax. For example, if you have a field in your incoming data set called uid, you could use something like:
  
  `https://demo.dev.data3sixty.com/api/v2/assets/${uid}`

  Doing so would insert the value found in the uid field for each incoming record into the record’s REST API call.

- **HTTP Method**
  The HTTP method to use. Options include: GET, PUT, POST, DELETE, and HEAD.

- **Authentication**
  The authentication type. Options include: None, Basic, Digest, or AWS. If required, a username and password may also be specified.

- **Call Type**

<table>
<thead>
<tr>
<th>Call Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>For basic calls that do not require paging.</td>
</tr>
<tr>
<td>Paging Call using Request Parameter</td>
<td>A paging call where a request parameter provides the offset or page number.</td>
</tr>
<tr>
<td>Paging Call using Response Marker</td>
<td>A paging call where there is a marker in the response that should be passed in to the next request.</td>
</tr>
</tbody>
</table>
■ Paging Variable Settings
  ■ If paging using a Request Parameter, Paging Variable Settings may be used to define a Name, Initial Value, and Increment Value. The Name specified must then be included in the URL or Request Body of the call as a substitutable variable using the ${} syntax.
  ■ If paging using a Response Marker, Paging Variable Settings may be used to define a Name. The Name specified must then be included in the URL or Request Body of the call as a substitutable variable using the ${} syntax. This variable is used to substitute in the value of the marker in the response.

Request Headers tab
This tab allows you to create a list of name/value pairs for the headers to pass to the REST service with each call. The value of a header can contain field value references using the ${} syntax.

Request Body tab
If the HTTP Method selected in the Request tab is of a type that takes a body, this tab will become enabled.
This tab allows you to define the following:
  ■ Content Type
    The content type of the request body. Defaults to application/json.
  ■ Body
    This is a text area that lets you enter text for a body. The body can contain field value references using the ${} syntax.

Response tab
This tab allows you to name the fields that will accept the contents of the REST call’s response.
These fields are as follows:
  ■ Status Field Name
    The name of the output field in which to return the response status. The type of this field is integer.
  ■ Body Field Name
    The name of the output field in which to return the response body if there is one. The type of this field is string.
  ■ Content Type Field Name
    The name of the output field in which to return the content type of the response if there is a response body. The type of this field is string.
Response Tab: Edit Paging Script
If either of the Paging Call Types is in use, an Edit Paging Script button becomes enabled in the Response Tab. This button allows you to create a script to control paging.

This script is passed an input object called `input` that contains a string field called `response`, which contains the response from the REST call. The script is also passed an output object called `output` that contains a boolean field called `continueIteration`, an integer field called `maxIterations`, and a string field called `nextMarker`. After script execution, if the `continueIteration` flag is set, then the node does the following:

- For the Response Marker Call Type, if the `nextMarker` value is set, another REST call is executed with the `restMarker`. The response of this new call is then passed into the script for further evaluation and the cycle continues until `continueIteration` or `nextMarker` is not set by the script.

- For the Request Parameter Call Type, if the `maxIterations` value is > 0, the system will perform as many calls as specified in `maxIterations`, with the paging variable incremented for each call. The script is not evaluated again. Note that `maxIterations` is inclusive of the call whose response is being processed.

- For the Request Parameter Call Type, if `maxIterations` is <=0 or not set, the system will increment the paging variable and execute another REST call. The response of this new call is then passed into the script for further evaluation and the cycle continues until `continueIteration` is not set by the script.

The script can use the JSON.parse Javascript function to parse the response string that is sent in.

Additionally, the script editor is prepopulated with the script required to process paginated responses from Govern REST calls when the Call Type is Request Parameter.

Proxy tab
This tab allows you to enable a proxy configuration.

It contains the following parameters:

- URL
  The url to the proxy server. The URL can contain field value references using the ${} syntax.

- Authentication
  The authentication type. Options include: None, Basic, or Digest. A username and password may also be specified in this section.
Check Nodes

Profile Data Node

The Profile Data node allows you to gather information about any field in your data set. Using this node generates a sheet with fields containing useful information about each profiled field. Fields generated by the Profile Data node are as follows.

Note: When building your Analysis, remember that profile information is representative of your data sample. To obtain an actual profile of the entire data set, you need to execute your Analysis.

field
The name of the field you are profiling.

totalCount
The total number of records in the field.

uniqueCount
The number of distinct values with only 1 instance.

distinctCount
The number of distinct records in the field. I.e., the number of records with a unique value.

duplicateCount
The number of records that have the same value.

distinctPercent
The percent of records in the field that are distinct. I.e., distinctCount/totalCount.

uniquePercent
The percent of records in the field that are unique. I.e., uniqueCount/totalCount.

duplicatePercent
The percent of records in the field that are duplicates. I.e., duplicateCount/totalCount.

completePercent
Represents what percentage of records in the field are not empty or not null. I.e., ((totalCount-(nullCount+emptyCount))/totalCount).

blankCount
The number of records which are blank strings, i.e., records that contain only whitespace.

minLength
Computes the minimum length of non-null and non-empty string fields. The minimum length of blank values is 0.
Using the Analysis Designer

Analysis Designer Nodes

**maxLength**
Computes the maximum length of non-null and non-empty string fields. The maximum length of blank values is 0.

**nullCount**
The number of records in the field that contain null values. Null values are those that equal '', NULLF(), or those where the value is empty. An empty value is one that contains only spaces and has a string length of 0.

**emptyCount**
The number of records that are null or have a string length of 0.

**max**
- Numeric fields: The maximum numeric value in the field.
- String fields: The string that appears at the end of the list, when all values in the field are sorted alphabetically.
- Date fields: The most recent date in the field.

Note: Null values are not evaluated by max profiling.

**min**
- Numeric fields: The minimum numeric value in the field.
- String fields: The string that appears at the beginning of the list, when all values in the field are sorted alphabetically.
- Date fields: The least recent date in the field.

Note: Null values are not evaluated by min profiling.

**values**
The **values** field generates a column where each record is an array containing all unique values for a profiled field, counts of each unique value, and percentages representing what percent a unique value’s record count is of the total record count.

For example, consider the following data set.

**values Data Set**

<table>
<thead>
<tr>
<th>field1</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
</tr>
<tr>
<td>two</td>
</tr>
<tr>
<td>three</td>
</tr>
<tr>
<td>three</td>
</tr>
<tr>
<td>four</td>
</tr>
</tbody>
</table>
Profiling field1 would produce the following record in the values field:
values = [{'value': 'one', 'two', 'three', 'four'}, {'totalCount': 1, 1, 2, 1}, {'valuePercent': 20, 20, 40, 20}]

When viewing the values field for an individual profiled field, you can also use the arrow navigation buttons to view the values fields for other profiled fields.

patterns
The patterns field generates a column where each record is an array containing all unique value patterns for a profiled field, counts of each unique value pattern, percentages of each unique value pattern, and a regex for each unique value pattern.

For example, consider the following data set.

patterns Data Set

<table>
<thead>
<tr>
<th>measure1</th>
<th>string2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>10</td>
<td>xy</td>
</tr>
<tr>
<td>100</td>
<td>xyz</td>
</tr>
</tbody>
</table>

Profiling measure1 would produce the following records in the patterns field:
pattern = [{'pattern': N, NN, NNN}, {'totalCount': 1, 1, 1}, {'patternPercent':33.33, 33.33, 33.33}, {'patternRegex': \d{1}, \d{2}, \d{3}]

Profiling string1 would produce the following records in the patterns field:
pattern = [{'pattern': a, aa, aaa}, {'totalCount': 1, 1, 1}, {'patternPercent':33.33, 33.33, 33.33}, {'patternRegex': \w{1}, \w{2}, \w{3}]

When viewing the pattern field for an individual profiled field, you can also use the arrow navigation buttons to view the patterns fields for other profiled fields.

Custom Counters
Custom Counters allow you to create expressions to apply to individual fields, which are used to create a count of values that satisfy the expression.

For example, consider the following data set.

Custom Counter Data Set

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>100</td>
</tr>
<tr>
<td>002</td>
<td>125</td>
</tr>
</tbody>
</table>
Were you to create a Custom Counter field named \textit{greaterThan150} using the expression \textit{value < 150}, \textit{greaterThan150} would equal 2.

\textit{Value Limit}

The Value Limit parameter controls how many values can be displayed in the Profile Data node’s \textit{values} and \textit{patterns} array fields.

For the \textit{values} and \textit{patterns} fields, if more values are found than the Value Limit, no values will be displayed.

By default, Value Limit is set to 10,000. This is the value that will be used if the parameter is left blank.

\textit{Completeness Check Node}

The Completeness Check node creates a single field containing a Boolean value, which represents whether a selected field has completely passed a set of checks.

If all checks pass, True is returned; if at least one check fails, False is returned.

\textit{Output Field Name}

This is the name of field that will hold the Boolean value representing completeness.

\textit{Field to Check}

This is the field from your data set that you’d like to check for completeness. Only one field may be checked per completeness check; however, multiple completeness checks can be created withing a single Completeness Check node.

\textit{Completeness Checks}

\begin{itemize}
  \item \textbf{Not Null}
    Returns True if value is not null; False if value is null.
  \item \textbf{Not Empty}
    This check returns True if a record value found in the Placeholder Field is not null or has a String length greater than 0, or False if a record value found in the Placeholder Field is null or has a String length of 0.
  \item \textbf{Not Blank}
    Returns True if value is not blank; False if value is blank.
    Values that are considered blank are those that are empty (as defined above in \textit{emptyCount}) or those that contain only spaces.
\end{itemize}
Using the Analysis Designer

**Static Text**
Enter a value to which to compare each record’s value. Returns True if Static Text and text in record match; returns False otherwise.

**Expression**
Allows you to write custom expressions to evaluate record values.
For example, if you were to write the expression $date = \text{DATE}(2005, 12, 1)$, and use it to evaluate the following data set, the listed results would be produced in the $\text{isFirstDate}$ field generated by the Completeness Check node.

### Completeness Check Expression Data Set

<table>
<thead>
<tr>
<th>date</th>
<th>isFirstDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005/12/02 00:00:00.0</td>
<td>False</td>
</tr>
<tr>
<td>2005/12/03 00:00:00.0</td>
<td>False</td>
</tr>
<tr>
<td>2005/12/01 00:00:00.0</td>
<td>True</td>
</tr>
</tbody>
</table>

**Consistency Check Node**
The Consistency Check node is used to evaluate whether a record’s values satisfy an expression or set of expressions. The node produces a new field containing Boolean values for each expression that is written. If a record satisfies the conditions in an expression, the Consistency Check node will output True. Conversely, when a record doesn’t satisfy an expression, the node will output False.

**Name**
When adding a new Consistency Check, **Name** is the name that will be given to the field containing the Consistency Check’s Boolean results.

**Expression**
**Expression** is the expression that each record will be evaluated against during the check.

**Consistency Check Example**
Suppose you had the following data set:

### Consistency Check Node Data Set

<table>
<thead>
<tr>
<th>account_number</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>007</td>
<td>1200</td>
</tr>
<tr>
<td>007</td>
<td>1000</td>
</tr>
<tr>
<td>007</td>
<td>999</td>
</tr>
</tbody>
</table>
6 Using the Analysis Designer

Analysis Designer Nodes

For this data set, we could write a Consistency Check with a Name of `007AboveOneThousand` and an Expression of `accountNumber = '007' && value > 1000`.

Doing so would produce the following data set:

**Consistency Check Node Results Data Set**

<table>
<thead>
<tr>
<th>account_number</th>
<th>value</th>
<th>007AboveOneThousand</th>
</tr>
</thead>
<tbody>
<tr>
<td>007</td>
<td>1200</td>
<td>True</td>
</tr>
<tr>
<td>007</td>
<td>1001</td>
<td>True</td>
</tr>
<tr>
<td>007</td>
<td>999</td>
<td>False</td>
</tr>
</tbody>
</table>

Note that multiple Consistency Checks may be written per Consistency Check node, so you could also create the following:

*Name: valueAboveOneThousand*

*Expression: value > 1000*

Which, in combination with `007AboveOneThousand`, would produce the following results.

**Consistency Check Node Results Data Set (two checks)**

<table>
<thead>
<tr>
<th>account_number</th>
<th>value</th>
<th>007AboveOneThousand</th>
<th>aboveOneThousand</th>
</tr>
</thead>
<tbody>
<tr>
<td>007</td>
<td>1200</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>007</td>
<td>1001</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>007</td>
<td>999</td>
<td>False</td>
<td>False</td>
</tr>
</tbody>
</table>

**Recon Check Node**

The Recon Check Node allows you to reconcile records from up to 4 sources, using a number of different validation options.

*Input Mapping Tab*

When inputs are connected to the Recon Check node, their Display Names and Field Names will appear within the Input Mapping tab. Source Field names may be changed within this tab, but note that they must be unique per Recon Check node (i.e., no two inputs may have the same name).

*Mapping Source Fields*

With the Map Source Fields button, you will need to select the fields you want to group by and map these fields between inputs.
For example, if you had 3 inputs and you wanted to perform a Recon check that grouped by an id field, you would need to select the id field from each input and map it to the id fields in all other inputs.

For example, let us assume that Input1, Input2, and Input3 contain the following fields and values:

**input1 Data Set**

<table>
<thead>
<tr>
<th>input1ID</th>
<th>measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>100</td>
</tr>
<tr>
<td>001</td>
<td>101</td>
</tr>
<tr>
<td>001</td>
<td>102</td>
</tr>
</tbody>
</table>

**input2 Data Set**

<table>
<thead>
<tr>
<th>input2ID</th>
<th>measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>100</td>
</tr>
<tr>
<td>001</td>
<td>101</td>
</tr>
<tr>
<td>001</td>
<td>120</td>
</tr>
</tbody>
</table>
Using the Analysis Designer

Analysis Designer Nodes

input3 Data Set

<table>
<thead>
<tr>
<th>input3ID</th>
<th>measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>100</td>
</tr>
<tr>
<td>001</td>
<td>101</td>
</tr>
<tr>
<td>001</td>
<td>102</td>
</tr>
</tbody>
</table>

Once we have grouped by and mapped id fields, the Recon Check node will restructure the 3 incoming data sets as follows, creating array fields for each input, grouped by the chosen grouping field:

Inputs in Recon Check Node

<table>
<thead>
<tr>
<th>input1ID</th>
<th>input1</th>
<th>input2</th>
<th>input3</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>[{measure=100}, {measure=101}, {measure=102}]</td>
<td>[{measure=100}, {measure=101}, {measure=102}]</td>
<td>[{measure=100}, {measure=120}, {measure=102}]</td>
</tr>
</tbody>
</table>

Validate Tab: Adding Recon Check Matches

Once fields are mapped and grouping fields are chosen, Recon Check Matches can be created to evaluate the values of records (contained in an array) associated to each group.

- **One to One**
  For every one value on the left there is one value on the right. Values need not match. For example, [{001:001}] and [{001:002}] are both one to one.

- **One to Many**
  For every one value on the left there are multiple values on the right. Values need not match. For example, [{001:001, 001}] and [{001:002, 003}] are both one to many.

- **Many to One**
  Multiple values on the left, one value on the right. Values need not match. For example, [{001, 001: 001}] and [{002, 003:001}] are both many to one.

- **Many to Many**
  Multiple values on the left, multiple values on the right. Values need not match. For example, [{001, 001: 001, 001}] and [{002, 003:004, 005}] are both many to many.

- **Accumulate**
  Used to accumulate a set of values on the left and to accumulate a set of values on the right, and then check to see if accumulations match.

Accumulate Example

Consider the following left and right inputs, where inputs have been mapped by id:
Left Data Set

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>100</td>
</tr>
<tr>
<td>001</td>
<td>200</td>
</tr>
<tr>
<td>001</td>
<td>300</td>
</tr>
</tbody>
</table>

Right Data Set

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>200</td>
</tr>
<tr>
<td>001</td>
<td>200</td>
</tr>
<tr>
<td>001</td>
<td>200</td>
</tr>
</tbody>
</table>

The following Accumulate Recon Checks could be created to produce results as shown.

1. **Left Field Name**: left  
   **Left Condition**: \{left\}.value = 200  
   **Left Expression**: \{left\}.value  
   **Left Computation**: mean  
   **Right Field Name**: right  
   **Right Condition**: None  
   **Right Expression**: \{right\}.value  
   **Right Computation**: mean  
   **Result**: True.

   On the left, the Recon node will find one value that matches the Left Condition. For that one value, it will use the Left Expression in a mean calculation, resulting in 200/1.

   On the right, the Recon node will find three values that match the Right Condition. For those three values, it will use the Right Expression in a mean calculation, resulting in 600/3.

   Since 200/1 = 600/3, the check evaluates to True.

**Match**

Used to match left and right values.  
**Match condition** allows you to specify a condition to match on. Only values that satisfy that condition will be considered when matching.  
**Match expression** allows you to specify an expression to manipulate values before match consideration is performed.
**Match Example**
Consider the following left and right inputs, where inputs have been mapped by id:

**Left Data Set**

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>100</td>
</tr>
<tr>
<td>001</td>
<td>200</td>
</tr>
<tr>
<td>001</td>
<td>300</td>
</tr>
</tbody>
</table>

**Right Data Set**

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>200</td>
</tr>
<tr>
<td>001</td>
<td>200</td>
</tr>
<tr>
<td>001</td>
<td>200</td>
</tr>
</tbody>
</table>

The following Match Recon Checks could be created to produce results as shown.

1. **Match Condition**: `{Left}.value = 200
   - Match Expression: `{Left}.value = {Right}.value
   - Result: True. Only the Left record with value = 200 will be considered, against all values on the right.

2. **Match Condition**: None
   - Match Expression: `{Left}.value = {Right}.value
   - Result: False. Since no Match condition is specified, all values on the left are compared to all values on the right - and not all of these values match.

3. **Match Condition**: `{Left}.value = 100
   - Match Expression: `{Left}.value * 2 = {Right}.value
   - Result: True. Only the Left record with value = 100 will be considered, against all values on the right. Before the left value is considered, however, the Match Expression multiplies it by 2, making it 200.

**Recon Check Match Name**
When creating new Recon Check Matches, you will need to assign them a name. This name will be the name of the field that contains Boolean results representing whether a record has passed the Recon Check. Records that return *True* are those that conform to the chosen Match Type; records that return *False* are records that do not.
**Timeliness Check Node**

The Timeliness Check Node is used to check whether the amount of time between two fields surpasses an acceptable level. The node produces a new field containing Boolean values for each Timeliness Check that is written. If the difference between the two chosen fields is less than or equal to the acceptable interval, the record is considered timely and a *True* value is returned. Conversely, if the difference between the two chosen fields is greater than the acceptable interval, the record is not considered timely and a *False* value is returned.

**Check Result Field Name**

This is the name of the Boolean field the node will produce for the Timeliness Check.

**Start Time Field**

This is the field containing start dates to be used in the Timeliness Check. Only Date, DateTime, or Time fields can be used.

**End Time Field**

This is the field containing end dates to be used in the Timeliness Check. Only Date, DateTime, or Time fields can be used.

**Allowed Time Difference**

This is the acceptable difference between the Start Time Field and the End Time Field to check for. If the difference between the Start Time Field and the End Time Field is less than or equal to the Allowed Time Difference, the Timeliness Check will return *True*; otherwise, it will return *False*.

**Timeliness Check Example**

Suppose you had the following data set:

<table>
<thead>
<tr>
<th>orderDate</th>
<th>shipDate</th>
</tr>
</thead>
<tbody>
<tr>
<td>08/08/2017</td>
<td>08/10/2017</td>
</tr>
<tr>
<td>08/09/2017</td>
<td>08/12/2017</td>
</tr>
<tr>
<td>08/10/2017</td>
<td>08/14/2017</td>
</tr>
</tbody>
</table>

For this data set, we could define a Timeliness Check with a Check Result Field Name of *ShippedOnTime*. For our Start Time Field we could use *orderDate*, and for our End Time Field we could use *shipDate*. For our Allowed Time Difference we could set an interval of 3 days, to check whether an order has shipped within 3 days of being placed.

Our result set would be as follows:
Using the Analysis Designer

Analysis Designer Nodes

Timeliness Check Node Results Data Set

<table>
<thead>
<tr>
<th>orderDate</th>
<th>shipDate</th>
<th>ShippedOnTime</th>
</tr>
</thead>
<tbody>
<tr>
<td>08/08/2017</td>
<td>08/10/2017</td>
<td>True</td>
</tr>
<tr>
<td>08/09/2017</td>
<td>08/12/2017</td>
<td>True</td>
</tr>
<tr>
<td>08/10/2017</td>
<td>08/14/2017</td>
<td>False</td>
</tr>
</tbody>
</table>

Note that, as with other Check nodes, multiple Timeliness Checks may be written per Timeliness Check node.

Value Conformity Node

The Value Conformity Node is used to check whether data set values match specific values or fall within a range of values. The node produces a new field containing Boolean values for each Value Conformity Check that is written. If a data set value matches any of the values in a Value Conformity Check or falls within the Check's range, a True value is returned. Conversely, if a data set value does not match any of the values in a Value Conformity Check or falls outside of the Check’s range, a False value is returned.

Output Field Name
This is the name of the Boolean field the node will produce for the Value Conformity Check.

Field
This is the field upon which to perform the Value Conformity Check. As discussed below, fields of different data types allow for different value matching options.

Numeric Checks

- **Value List**
  Check to see if a value matches any values from a set of values. Values in the set may be added one at a time using the New Value option or as a comma separated list with the Multiple Values option.

- **Range**
  Check to see if a value falls within a range of values. From values are included in the range; To values are not.

Date, DateTime, and Time Checks

- **Value List**
  Check to see if a value matches any values from a set of values. Values in the set may be added one at a time using the New Value option.
Range
Check to see if a value falls within a range of values. *From* values are included in the range; *To* values are not.

String Checks

Value List
Check to see if a value matches any values from a set of values. Values in the set may be added one at a time using the *New Value* option or as a comma separated list with the *Multiple Values* option.

Note that when using the *Multiple Values* option for Strings, any spaces between comma separated values will be interpreted literally. For example, suppose you wanted to add the following values:

these, are, my, values

If you were to add these values to the list as shown above, with spaces, the spaces would be added to the beginning of the String that they precede, effectively adding the following values to your check (quotes shown to emphasize spaces):

‘these’, ‘ are’, ‘ my’, ‘ values’

To check for your actual values, you would want to add them to the list without spaces, as follows:

these,are,my,values

Type Conformity Node

The Type Conformity Node is used to check whether data set values conform to patterns. The node produces a new field containing Boolean values for each Type Conformity Check that is written. If a data set value matches the pattern in a Type Conformity Check, a *True* value is returned. Conversely, if a data set value does not match the pattern in a Type Conformity Check, a *False* value is returned.

Output Field Name
This is the name of the Boolean field the node will produce for the Type Conformity Check.

Field to Check
This is the field upon which to perform the Type Conformity Check. As discussed below, fields of different data types allow for different pattern matching options.
6 ■ Using the Analysis Designer

Analysis Designer Nodes

Type Conformity Checks

- **Regular Expression**
  This option allows you to verify whether a field’s value conforms to a specific Java Regular Expression Pattern. Any Java Regex Pattern can be used.

- **Number**
  This option allows you to verify whether a field’s value conforms to a specific number formatting pattern. Either predefined formatting patterns or a custom pattern can be used.

- **Date**
  This option allows you to verify whether a field’s value conforms to a specific date formatting pattern. Either predefined formatting patterns or a custom pattern can be used.

Execute Rule Library

The Execute Rule Library node allows you to apply the reusable rules you have created within a Rule Library to a data set you are working on in an Analysis. The output of this node is a data set containing the fields under evaluation, a result field, and an error field.

Details tab

- **Rule Library**
  This is the Rule Library that you have already created, which contains the reusable rules you want to apply to your data set.

- **Rule Group**
  Within the selected Rule Library, this is the Data Quality Rule Group that contains the reusable rules that you want to execute.

- **Result Field**
  This parameter allows you to give a name to the result field that will be produced by the Execute Rule Library node and will contain a Boolean result. The result for each evaluated record - True or False - is the result of ANDing every reusable rule applied to the record. In other words, to produce a True result, an evaluated record must cause every reusable rule being executed by the node to evaluate to True.

- **Error Reason Field**
  This parameter allows you to give a name to the error field that will be produced by the Execute Rule Library node. If a record’s Result Field evaluates to False, the Error Reason field will contain the list of reusable rules which evaluated to False. If a record’s Result Field evaluates to True, the Error Reason field will be empty.

- **Execute All Rules in a Group**
  Check this box if you would like to apply every reusable rule from the selected Data Quality Rule Group to the data set.
■ Rules to Execute
If you have unchecked *Execute All Rules in a Group*, you can select individual reusable rules from the Data Quality Rule Group to use.

**Placeholder Mapping tab**
The Placeholder Mapping tab is used to map incoming fields to the Placeholder Fields within the selected Data Quality Rule Group. Incoming fields that are mapped to Placeholder Fields will be evaluated in the selected reusable rules wherever the Placeholder Fields are referenced, in the rules' definitions.

**Profile Data Legacy Node**
***Note this node is deprecated and has been replaced by the Profile Data node***
The Profile Data node allows you to gather information about any field in your data set. Using this node generates a sheet with fields containing useful information about each profiled field. Fields generated by the Profile Data node are as follows.

Note: When building your Analysis, remember that profile information is representative of your data sample. To obtain an actual profile of the entire data set, you need to execute your Analysis.

- **field**
The name of the field you are profiling.
- **uniqueCount**
The number of unique values found in the field.
- **nullsCount**
The number of records in the field that contain null values. Null values are those that equal ‘’, NULLF(), or those where the value is empty.
- **emptyCount**
The number of records that are null or have a string length of 0.
- **max**
  - Numeric fields: The maximum numeric value in the field.
  - String fields: The string that appears at the end of the list, when all values in the field are sorted alphabetically.
  - Date fields: The most recent date in the field.

Note: Null values are not evaluated by max profiling.
Using the Analysis Designer

Analysis Designer Nodes

**min**
- Numeric fields: The minimum numeric value in the field.
- String fields: The string that appears at the beginning of the list, when all values in the field are sorted alphabetically.
- Date fields: The least recent date in the field.

Note: Null values are not evaluated by min profiling.

**values**
The *values* field generates a column where each record is an array containing all unique values for a profiled field and counts of each unique value.

For example, consider the following data set.

**values Data Set**

<table>
<thead>
<tr>
<th>field1</th>
</tr>
</thead>
<tbody>
<tr>
<td>one</td>
</tr>
<tr>
<td>two</td>
</tr>
<tr>
<td>three</td>
</tr>
<tr>
<td>three</td>
</tr>
<tr>
<td>four</td>
</tr>
</tbody>
</table>

Profiling *field1* would produce the following record in the *values* field:

```
values = [{'value': 'one', 'two', 'three', 'four'}, {'totalCount': 1, 1, 2, 1}]
```

When viewing the *values* field for an individual profiled field, you can also use the arrow navigation buttons to view the *values* fields for other profiled fields.

**patterns**
The *patterns* field generates a column where each record is an array containing all unique value patterns for a profiled field and counts of each unique value pattern.

For example, consider the following data set.

**patterns Data Set**

<table>
<thead>
<tr>
<th>measure1</th>
<th>string2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
</tr>
<tr>
<td>10</td>
<td>xy</td>
</tr>
<tr>
<td>100</td>
<td>xyz</td>
</tr>
</tbody>
</table>
Using the Analysis Designer

Analysis Designer Nodes

Profiling `measure1` would produce the following records in the `patterns` field:

```
pattern = [{'pattern': N, NN, NNN}, {'totalCount': 1, 1, 1}]
```

Profiling `string1` would produce the following records in the `patterns` field:

```
pattern = [{'pattern': a, aa, aaa}, {'totalCount': 1, 1, 1}]
```

When viewing the `pattern` field for an individual profiled field, you can also use the arrow navigation buttons to view the `patterns` fields for other profiled fields.

**Custom Counters**

Custom Counters allow you to create expressions to apply to individual fields, which are used to create a count of values that satisfy the expression.

For example, consider the following data set.

**Custom Counter Data Set**

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>100</td>
</tr>
<tr>
<td>002</td>
<td>125</td>
</tr>
<tr>
<td>003</td>
<td>150</td>
</tr>
<tr>
<td>004</td>
<td>175</td>
</tr>
<tr>
<td>005</td>
<td>200</td>
</tr>
</tbody>
</table>

Were you to create a Custom Counter field named `greaterThan150` using the expression `value < 150`, `greaterThan150` would equal 2.

**Value Limit**

The Value Limit parameter controls how many values can be displayed in the Profile Data node’s `values` and `patterns` array fields.

For the `values` and `patterns` fields, if more values are found than the Value Limit, no values will be displayed.

By default, Value Limit is set to 10,000. This is the value that will be used if the parameter is left blank.
6 - Using the Analysis Designer

Analysis Designer Nodes

Analytics Nodes

Analytic Models

If you intend to use any of the Analytics nodes available in Infogix Data3Sixty’s Analysis Designer, you will need to create Analytic Models first. Analytic Models allow you to Train, Score, and Evaluate data sets, via the Analytics nodes; and, there is one Analytic Model type available for each type of Analytic node.

How to use an Analytic Model

The specific application of an Analytic Model is completely dependent on the type of data you’d like to Train, Score, and Evaluate; however the general work flow of using Analytic Models with Analytics nodes is as follows.

Create an Analytic Model

You’ll first want to find an appropriate Pipeline and Path, and then create a blank Analytic Model that matches the type of Analytics node you want to use in an Analysis. This Analytic Model will hold child training models that are created each time an Analytics node using the Train operation executes or is tested within an Analysis. Child training models can then be used to Score new data sets in other Analyses.

Choose Training Data

Regardless of which Analytics node is used, Training Data needs:

- Input fields/parameters: i.e., attributes about each record.
- A Label field: i.e., the answer that the model will learn to associate with input parameters.

Choose Validation Data

If you intend to create multiple child models and compare them using the Evaluate operation, you will also need to set aside a Validation Data Set. Like the Training Set, the Validation Set also needs a Label Field - that is, a field containing an actual value that can be compared to a predicted value. The Validation Data Set will also need the same set of fields that was used to generate the trained model, so that Evaluation can take values from these fields, generate predictions, compare predictions to actual values in the Label field, and ultimately inform you on the quality of your model using the Evaluation Metrics described below.

Generating Training and Validation Data Sets

Training and Validation Data Sets are typically generated by splitting an initial labeled data set - that is, a data set where the answers are known - into two subsets. For Training, a subset that is ~60 - 70% of the original data set is used; for Validation, the remaining ~30-40% is used.
Currently, there is no built in way to generate this split using Infogix Data3Sixty. The split can however be performed using an Analysis as shown below*.

Splitting into Training and Validation Sets

With this method, 1) an Autonumber node is first used to give each record in the data set a unique id (note that if records already have unique ids, this step is not necessary). After that, 2) a Sample node is used to pull 60% from the Data Store Input. Finally, 3) the Not In node is used to find IDs that are present in the entire data set, but not present in the 60% sample. The result of the Not In node is then pushed to the Validation Data Set, and the 60% Sample is pushed to the Training Data Set.

*Depending on the data you have available to you, it may be easier to just generate the Training and Validation sets outside of Infogix Data3Sixty, and then simply create a Data Store for each set.

**Train Models**

Once you have created an Analytic Model, you can use it to Train with an Analytics node in the Analysis Designer.

To do so, you'll need to identify an appropriate Data Store Input to use for training, and then connect it to the chosen Analytics node - either directly or by way of Enhance nodes.
After that, you’ll need to select the Analytic Model you created to store child training models, in the node’s Properties panel, and use the Train operation. Depending on the Analytics node in use, other properties will also need to be selected. Once these selections are made, the data will be ready to be trained.

At this point, you will be able to see the effects of training in the Analysis by clicking Accept or Test; and, your new Prediction field will appear in the Analytics node’s sheet. This Prediction Field, along with other fields, can then be pushed to a Data Store Output to be used by other Data Stages.

Once the Analysis is Saved and Executed, a child model will be stored in your Analytic Model, which Infogix Data3Sixty will name with a date and time.

**Re-Training and Creating Child Models**

Each time an Analysis containing an Analytics node is tested or executed, it will create a training model and store it in the chosen Analytic Model. If the same Analytic Model is used for multiple training runs, new training models are stored as **Child Models**.
By default, Infogix Data3Sixty names each child model with a date and time; however, to select a child model for scoring, you need to give it a display name.

Each time you Train a new child model, it will only use the data that was input to the Analytics node during the Analysis run. Child models that are part of the same Analytic Model are independent, and do not build off of one another in any way.

This means that when Re-Training, you can improve your Analytic Model by Training with: a) different data and/or b) different input parameters.

**Evaluating Models**

Once you have trained models, you can evaluate them. To do so, input a Validation set into an Analytics node.

After that, you’ll need to select the Analytic Model that contains the child models you want to evaluate, in the Analytics node’s Properties panel, and use the Evaluate operation.
You can choose one or multiple child models from your Analytic Model and Infogix Data3Sixty will rank their predictive accuracy. Knowing which child models are the most accurate can help you determine both: 1) the accuracy of any scoring you’ve already performed and 2) which models to use for future scoring.

**Measures used for Evaluation**

When an Evaluation runs, it produces measures to assess the performance of your Child Models. Below you can find an overview of the measures used for each type of Analytics node.

**Regression or Recommendation: Root Mean Square Error (RMSE)**

\[
RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2}
\]

The Root Mean Square Error is a measure used to evaluate Regression or Recommendation models. RMSE is the square root of the mean of the square of the summation of all errors between predicted values and labeled values.

In general, the lower the RMSE, the better the performance of a model. What typifies a “low” RMSE depends on the range of values in the model’s label field.
Using the Analysis Designer

Classification Evaluation Metrics

When Training a Classification Model using the Automatically Tune Parameters option, you’ll need to select an Evaluation Metric to use to rank the model when you compare it to others. A brief explanation of each available metric is provided below.

Classification: F Measure

\[ F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \]

F Measure is used to evaluate Classification models. F Measure uses a result set’s Precision and Recall to produce a value between 0 and 1 to represent the performance of a Classification model.

A result set’s **Precision** is the number of true positives \((tp)\) over all positives returned \((pr)\), or \(tp/pr\).

A result set’s **Recall** is the number of true positives \((tp)\) over actual positives \((ap)\) in the result set, or \(tp/ap\).

With F Measure, the closer to 1, the better the performance of the Classification model.

As a simple example of F Measure, suppose you had 10 items known to be in Class A \((ap = 10)\). After training, your model classifies 7 items as Class A \((pr = 7)\). After further investigation, however, you find you that only 5 of these items are actually supposed to be in Class A \((tp = 5)\).

In this example, your Precision would be \(tp/pr\), or 5/7. Your Recall, on the other hand, would be \(tp/ap\), only 5/10.

Using the formula above, the model’s F Measure would be roughly 0.59, suggesting that the model has room for improvement.

Profit

The Profit Evaluation Metric can help you find an optimal probability threshold for classification models. It does so by allowing you to specify dollar amounts for the profit gained from each True Positive and the cost incurred from each False Negative. These values can be placed in the **True Positive Profit** and **False Negative Cost** parameters, respectively.
When using the Profit Evaluation Metric, you will also need to specify a Setup Cost, which represents the cost incurred from setting up the model to get a more accurate measure of profit.

**Accuracy**

\[
\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total number of cases examined}}
\]

Accuracy is defined as the proportion of True results among the total number of cases examined. True Results includes both True Positives and True Negatives.

How to Interpret:
The closer to 1, the better.

**Sensitivity**

\[
\text{Sensitivity} = \frac{\text{True Positives}}{\text{Total Positives in set}}
\]

Sensitivity - also known as Recall - is defined as the True Positive Rate; or, the number of results which tested positive and are actually positive divided by the total number of positives in the set.

How to Interpret:
The closer to 1, the better.

**Specificity**

\[
\text{Specificity} = \frac{\text{True Negatives}}{\text{Total Negatives in set}}
\]

Specificity is defined as the True Negative Rate; or, the number of results which tested negative and are actually negative divided by the total number of negatives in the set.

How to Interpret:
The closer to 1, the better.

**Precision**

\[
\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}
\]

Precision is defined as the number of True Positives divided by the number of results that tested positive.

How to Interpret:
The closer to 1, the better. I.e., the fewer the False Positives, the better the model.

**Negative Predictive Value**

\[
\text{Negative Predictive Value} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Negatives}}
\]

Negative Predictive Value is defined as the number of True Negatives divided by the number of results that tested negative.

How to Interpret:
The closer to 1, the better. I.e., the fewer the False Negatives, the better the model.

**False Positive Rate**

\[
\frac{\text{False Positives}}{\text{Total Negatives in set}} = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}
\]

False Positive Rate is defined as the number of False Positives divided by the total number of negatives in the set. Note that Total Negatives is equivalent to the sum of False Positives and True Negatives.

How to Interpret:

The closer to 0, the better. I.e., the fewer the False Positives, the better the model.

**False Discovery Rate**

\[
\frac{\text{False Positives}}{\text{False Positives} + \text{True Positives}}
\]

False Discovery Rate is defined as the number of False Positives divided by the number of results that tested positive in the set.

How to Interpret:

The closer to 0, the better. I.e., the fewer the False Positives, the better the model.

**False Negative Rate**

\[
\frac{\text{False Negatives}}{\text{Total Positives}} = \frac{\text{False Negatives}}{\text{False Negatives} + \text{True Positives}}
\]

False Negative Rate is defined as the number of False Negatives divided by the total number of positives in the set. Note that Total Positives is equivalent to the sum of False Negatives and True Positives.

How to Interpret:

The closer to 0, the better. I.e., the fewer the False Negatives, the better the model.

**Feature Importance**

Rather than ranking the model, this metric provides information about the impact each feature has on the model, where *features* are the fields that were selected as inputs during training. Features that have greater impact on the model have values closer to 1. Features that have no impact on the model are not listed.

**Choosing Scoring Data**

Regardless of which Analytics node is used, Scoring Data needs:

- The same set of Input parameter fields that were used to create the Scoring model.
- No Label field: The point of scoring is to predict on unlabeled data.
6 Using the Analysis Designer

**Analysis Designer Nodes**

**Scoring Data**

Once you have one or more child training models for a data set, you can enter the Analytic Model where they reside and choose one to use for Scoring.

After a scoring model is chosen, you can use an Analytics node to Score data—that is, to compare a new data set to a trained model. To do so, you’ll need to once again input an appropriate Data Store into an Analytics node.

After that, you’ll need to select the Analytic Model that contains your scoring model, in the node’s Properties panel, and use the Score operation. Depending on the Analytics node in use, other properties will also need to be selected. Once these selections are made, the data will be ready to be scored.

At this point, you will be able to see the effects of scoring in the Analysis by clicking Accept or Test; and, your new Prediction field will appear in the Analytics node’s sheet. This Prediction Field, along with other fields, can then be pushed to a Data Store Output to be used by other Data Stages— for example, a Data View that allows for the visualization of predictive analytics.
Once the Analysis is then Saved and Executed, scored data will be pushed to the Data Store Output. The accuracy of your scoring will be dependent on the strength of your scoring model.

**Segmentation**

The Segmentation node uses clustering algorithms to perform “unsupervised” learning. Unsupervised learning takes raw, unlabeled data and clusters it into segments. Data points that have been clustered into the same segment are likely to be related in some way.

In general, the Segmentation node can be useful if you have a data set to which you would like to apply discrete categories.

For example, given a data set containing customer demographics and purchase records, you could use the Segmentation node to derive market segments.

**Segmentation Training**

To perform segmentation training, you will need an unlabeled data set - that is, one where records have not been previously categorized. This data set will need to contain a set of fields, or parameters, that the Analytic Model can compare to create Segments.

**Number of Desired Segments**

Note that Segment numbering begins with 0; so, for example, if you wanted 4 segments you would actually need to specify 3.

**Loan Example: Training**
6 ■ Using the Analysis Designer

Analysis Designer Nodes

As a simple example of Segmentation training, consider a sample from a Loan data set. Suppose that the object of training is to group loan applicants into customer segments based on parameterized information about the applicant. (See Loan Segmentation Data Set 1).

Field names: amt = Loan Amount; yEm = Years Employed; dti = Debt to Income Ratio; inc = Annual Income; intR = Interest Rate; term = Loan Term; inst = Monthly installment; grade = Loan Grade.

Loan Segmentation Data Set 1

<table>
<thead>
<tr>
<th>id</th>
<th>amt</th>
<th>yEm</th>
<th>dti</th>
<th>inc</th>
<th>intR</th>
<th>term</th>
<th>inst</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>5000</td>
<td>10</td>
<td>20.97</td>
<td>80000</td>
<td>10.99</td>
<td>36</td>
<td>163.67</td>
<td>B</td>
</tr>
<tr>
<td>002</td>
<td>5000</td>
<td>3</td>
<td>24</td>
<td>28800</td>
<td>13.33</td>
<td>36</td>
<td>169.27</td>
<td>C</td>
</tr>
<tr>
<td>003</td>
<td>15000</td>
<td>10</td>
<td>10.81</td>
<td>75000</td>
<td>12.29</td>
<td>36</td>
<td>500.3</td>
<td>C</td>
</tr>
<tr>
<td>004</td>
<td>24000</td>
<td>9</td>
<td>23.76</td>
<td>87818</td>
<td>19.99</td>
<td>60</td>
<td>635.72</td>
<td>E</td>
</tr>
<tr>
<td>005</td>
<td>13200</td>
<td>10</td>
<td>24.05</td>
<td>70000</td>
<td>9.17</td>
<td>60</td>
<td>275.11</td>
<td>B</td>
</tr>
<tr>
<td>006</td>
<td>19000</td>
<td>10</td>
<td>13.12</td>
<td>56900</td>
<td>17.86</td>
<td>60</td>
<td>481.03</td>
<td>D</td>
</tr>
<tr>
<td>007</td>
<td>14500</td>
<td>5</td>
<td>24.25</td>
<td>63500</td>
<td>9.17</td>
<td>60</td>
<td>302.2</td>
<td>B</td>
</tr>
<tr>
<td>008</td>
<td>35000</td>
<td>2</td>
<td>30.61</td>
<td>84000</td>
<td>18.25</td>
<td>60</td>
<td>893.54</td>
<td>E</td>
</tr>
<tr>
<td>009</td>
<td>25000</td>
<td>10</td>
<td>12.02</td>
<td>110000</td>
<td>7.89</td>
<td>60</td>
<td>505.6</td>
<td>A</td>
</tr>
<tr>
<td>010</td>
<td>18000</td>
<td>1</td>
<td>21.46</td>
<td>70000</td>
<td>13.99</td>
<td>36</td>
<td>615.11</td>
<td>C</td>
</tr>
</tbody>
</table>

Specify Inputs

For training, 8 parameters could be used as Input Fields: amt, yEm, dti, inc, intR, term, inst, and grade.

Note that parameters that are not specified as Inputs to the Analytic Model (here, id) will still be passed through the node attached to records.

Create Prediction Field

After specifying Inputs, you would then need to create a Prediction Field with an appropriate name. Here we’ll use: CustomerSegment (CS).

We could then use the Segmentation tab in the node’s Properties panel to specify 4 segments - which would actually give us 5 segments (0,1,2,3 and 4).

After running this Analysis, the Segmentation node would output a data set containing all the specified Inputs, the Prediction Field containing Segment number, and any other fields attached to records. (See Loan Segmentation Data Set 2)
Importantly, training will also create a Child Model within the selected Analytic Model. This Child Model could later be used for scoring.

**Loan Segmentation Data Set 2**

<table>
<thead>
<tr>
<th>id</th>
<th>amt</th>
<th>yEm</th>
<th>dti</th>
<th>inc</th>
<th>intR</th>
<th>term</th>
<th>inst</th>
<th>grade</th>
<th>CS</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>5000</td>
<td>10</td>
<td>20.97</td>
<td>80000</td>
<td>10.99</td>
<td>36</td>
<td>163.67</td>
<td>B</td>
<td>3</td>
</tr>
<tr>
<td>002</td>
<td>5000</td>
<td>3</td>
<td>24</td>
<td>28800</td>
<td>13.33</td>
<td>36</td>
<td>169.27</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>003</td>
<td>15000</td>
<td>10</td>
<td>10.81</td>
<td>75000</td>
<td>12.29</td>
<td>36</td>
<td>500.3</td>
<td>C</td>
<td>0</td>
</tr>
<tr>
<td>004</td>
<td>24000</td>
<td>9</td>
<td>23.76</td>
<td>87818</td>
<td>19.99</td>
<td>60</td>
<td>635.72</td>
<td>E</td>
<td>4</td>
</tr>
<tr>
<td>005</td>
<td>13200</td>
<td>10</td>
<td>24.05</td>
<td>70000</td>
<td>9.17</td>
<td>60</td>
<td>275.11</td>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>006</td>
<td>19000</td>
<td>10</td>
<td>13.12</td>
<td>56900</td>
<td>17.86</td>
<td>60</td>
<td>481.03</td>
<td>D</td>
<td>4</td>
</tr>
<tr>
<td>007</td>
<td>14500</td>
<td>5</td>
<td>24.25</td>
<td>63500</td>
<td>9.17</td>
<td>60</td>
<td>302.2</td>
<td>B</td>
<td>4</td>
</tr>
<tr>
<td>008</td>
<td>35000</td>
<td>2</td>
<td>30.61</td>
<td>84000</td>
<td>18.25</td>
<td>60</td>
<td>893.54</td>
<td>E</td>
<td>4</td>
</tr>
<tr>
<td>009</td>
<td>25000</td>
<td>10</td>
<td>8.62</td>
<td>110000</td>
<td>7.89</td>
<td>60</td>
<td>505.6</td>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>010</td>
<td>18000</td>
<td>1</td>
<td>21.46</td>
<td>70000</td>
<td>13.99</td>
<td>36</td>
<td>615.11</td>
<td>C</td>
<td>2</td>
</tr>
</tbody>
</table>

*Segmentation: Scoring*

Once you have selected a Child Model within your Analytic Model to use for scoring, you can create another Analysis that uses a Segmentation node to score a new data set.

To do so, you will need a data set containing the same fields that were used as parameters when the scoring model was trained. During scoring, the Segmentation node will compare the values in these fields to values in the scoring model, to assign segments.

**Loan Example: Scoring**

To continue the Loan example from above, we would simply need another data set with the same parameters. This data set could be used as an input to a Segmentation node in an Analysis. (This Data Set is not listed).

To score, you would then need to select the Score operation and the Analytic Model that was used in training. You would also need to create a Prediction Field. For simplicity, you can give the Prediction field the same name that was used in training. In this example, that was: CustomerSegment (CS).

After running the scoring Analysis, the Segmentation node would output a data set containing the parameter fields, the Prediction Field, and any other field attached to records. This data set could then be further explored in Dashboards or the Infogix Data3Sixty Visualizer.
6 ■ Using the Analysis Designer

Analysis Designer Nodes

Segmentation: Evaluation
The Segmentation node does not feature an Evaluate operation, however the results of training/scoring can still be evaluated. To do so, use the Data Store Output from a training or scoring run to build a Data View.

Loan Example: Evaluation
In the case of the Loan Example outlined above, placing the Data Store Output of training or scoring into a Data View would allow you to analyze loan applicant segments. Such a Data View would enable Dashboards that visualize the parameters describing applicants, showing which applicants have similar parameter values. These applicants would be grouped into segments, or “clusters”.

Classification
The Classification node performs “supervised” learning algorithms to place a new observation into a preexisting, discrete category. Classification categories are derived from a training set of data that has already been labeled, or “clustered.” For this reason, the Classification node can be considered the complement of the Segmentation node.

Classification works best when attempting to output discrete values.
For example, given a labeled data set that matches customer demographics to loan types customers qualify for, you could apply the classification node to a new customer data set containing only demographic information to attempt to determine what types of loans customers will qualify for.
Classification: Training

To perform Classification training, you will need a labeled data set - that is, one where there is a field that already classifies records. This data set also needs to contain a set of fields, or parameters, that the Analytic Model can associate with classes.

Iris Example: Training

As a simple example of Classification, consider a sample from the famous “Iris Flower Data set.” Suppose that the object of training is to classify flower records into one of three Species based on collected measurements.

Iris Data Set 1

<table>
<thead>
<tr>
<th>SepalLength</th>
<th>SepalWidth</th>
<th>PetalLength</th>
<th>PetalWidth</th>
<th>Species</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>3.3</td>
<td>1.7</td>
<td>0.5</td>
<td>setosa</td>
</tr>
<tr>
<td>7.7</td>
<td>2.6</td>
<td>6.9</td>
<td>2.3</td>
<td>virginica</td>
</tr>
<tr>
<td>6.6</td>
<td>2.9</td>
<td>4.6</td>
<td>1.3</td>
<td>versicolor</td>
</tr>
<tr>
<td>4.4</td>
<td>3</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>5</td>
<td>3.5</td>
<td>1.3</td>
<td>0.3</td>
<td>setosa</td>
</tr>
<tr>
<td>6.4</td>
<td>2.7</td>
<td>5.3</td>
<td>1.9</td>
<td>virginica</td>
</tr>
<tr>
<td>5.7</td>
<td>2.5</td>
<td>5</td>
<td>2</td>
<td>virginica</td>
</tr>
<tr>
<td>6.4</td>
<td>3.1</td>
<td>5.5</td>
<td>1.8</td>
<td>virginica</td>
</tr>
<tr>
<td>6.5</td>
<td>3.2</td>
<td>5.1</td>
<td>2</td>
<td>virginica</td>
</tr>
<tr>
<td>5.4</td>
<td>3.4</td>
<td>1.7</td>
<td>0.2</td>
<td>setosa</td>
</tr>
</tbody>
</table>

This data set contains 4 parameters - SepalLength, SepalWidth, PetalLength and PetalWidth - and a Label field, Species. To Train an Analytic Model with this data set, you would specify the 4 parameters as Input Fields and Species as the Label Field.

You’d then need to create a Prediction Field with an appropriate name and the same data type as your Label Field. In this case, a field named SpeciesPred of the String data type will work.

After running your Analysis, the Classification node would output a data set containing all specified Inputs, the Label Field, the Prediction Field, and any other fields associated to records. (See Iris Data Set 2).
Using the Analysis Designer

Analysis Designer Nodes

Importantly, training will also create a child model within the selected Analytic Model. This child model could later be used for scoring.

Iris Data Set 2

<table>
<thead>
<tr>
<th>SepalLength</th>
<th>SepalWidth</th>
<th>PetalLength</th>
<th>PetalWidth</th>
<th>Species</th>
<th>SpeciesPrediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>3.3</td>
<td>1.7</td>
<td>0.5</td>
<td>setosa</td>
<td>setosa</td>
</tr>
<tr>
<td>7.7</td>
<td>2.6</td>
<td>6.9</td>
<td>2.3</td>
<td>virginica</td>
<td>virginica</td>
</tr>
<tr>
<td>6.6</td>
<td>2.9</td>
<td>4.6</td>
<td>1.3</td>
<td>versicolor</td>
<td>versicolor</td>
</tr>
<tr>
<td>4.4</td>
<td>3.0</td>
<td>1.3</td>
<td>0.2</td>
<td>setosa</td>
<td>setosa</td>
</tr>
<tr>
<td>5.0</td>
<td>3.5</td>
<td>1.3</td>
<td>0.3</td>
<td>setosa</td>
<td>setosa</td>
</tr>
<tr>
<td>6.4</td>
<td>2.7</td>
<td>5.3</td>
<td>1.9</td>
<td>virginica</td>
<td>virginica</td>
</tr>
<tr>
<td>5.7</td>
<td>2.5</td>
<td>5.0</td>
<td>2.0</td>
<td>virginica</td>
<td>virginica</td>
</tr>
<tr>
<td>6.4</td>
<td>3.1</td>
<td>5.5</td>
<td>1.8</td>
<td>virginica</td>
<td>virginica</td>
</tr>
<tr>
<td>6.5</td>
<td>3.2</td>
<td>5.1</td>
<td>2.0</td>
<td>virginica</td>
<td>virginica</td>
</tr>
<tr>
<td>5.4</td>
<td>3.4</td>
<td>1.7</td>
<td>0.2</td>
<td>setosa</td>
<td>setosa</td>
</tr>
</tbody>
</table>

Classification: Evaluation
To evaluate the accuracy of an Analytic Model - and by extension the accuracy of scoring that is performed using that model - you can use the Classification node’s Evaluate operation.
To evaluate a child training model, you need to use a Validation dataset as an input to a Classification node.

Iris Example: Evaluating the Child Training Model
To evaluate the child model created during the Iris data set training, you would need to input a Validation data set into a Classification node and select the Evaluate operation. The Evaluation would then produce an F-Measure, as well a number of other Evaluation Metrics not shown here.

Evaluation Metrics

<table>
<thead>
<tr>
<th>ModelDisplayName</th>
<th>ChildModelDisplayName</th>
<th>Rank</th>
<th>FMeasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Classification Model</td>
<td>Child Model 1</td>
<td>1</td>
<td>0.986</td>
</tr>
</tbody>
</table>
**Classification: Re-Training**

After Training and Evaluating your first child model, you can always choose to Train another one, in order to obtain better Evaluation Metrics and more accurate scoring results.

To retrain, you can use new data and/or different parameters as input fields. Each time you Re-Train using the same Analytic Model, another “child model” will be produced.

**Classification: Scoring**

Once you have selected a child model within your Analytic Model to use for scoring, you can create another Analysis that uses a Classification node to score an unlabeled data set - that is, to predict what class each record within the data set belongs to.

To do so, you will need a data set containing the same fields that were used as parameters when the scoring model was trained. During scoring, the Classification node will compare the values in these fields to values in the scoring model, to predict a class.

**Iris Example: Scoring Unlabeled Data**

To continue the Iris Example, consider an unlabeled data set that only contains values for the 4 parameters: SepalLength, SepalWidth, PetalLength and PetalWidth. (See Iris Data Set 3).

**Iris Data Set 3**

<table>
<thead>
<tr>
<th>SepalLength</th>
<th>SepalWidth</th>
<th>PetalLength</th>
<th>PetalWidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.7</td>
<td>2.8</td>
<td>6.7</td>
<td>2</td>
</tr>
<tr>
<td>5.2</td>
<td>3.4</td>
<td>1.4</td>
<td>0.2</td>
</tr>
<tr>
<td>6.1</td>
<td>3</td>
<td>4.9</td>
<td>1.8</td>
</tr>
<tr>
<td>5</td>
<td>3.5</td>
<td>1.3</td>
<td>0.3</td>
</tr>
<tr>
<td>5.4</td>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
</tr>
<tr>
<td>4.9</td>
<td>2.5</td>
<td>4.5</td>
<td>1.7</td>
</tr>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.3</td>
</tr>
<tr>
<td>5.8</td>
<td>2.6</td>
<td>4</td>
<td>1.2</td>
</tr>
<tr>
<td>5.9</td>
<td>3</td>
<td>4.2</td>
<td>1.5</td>
</tr>
<tr>
<td>6.9</td>
<td>3.1</td>
<td>5.4</td>
<td>2.1</td>
</tr>
</tbody>
</table>

To Score this data set, you would need to input it into a Classification node and select the Score operation. You’d then need to create a Prediction Field. Once again, a Prediction field named SpeciesPred of the String data type would suffice.
After running your Analysis, the Classification node would output a data set containing the parameter fields and the Prediction Field. (See Iris Data Set 4).

**Iris Data Set 4**

<table>
<thead>
<tr>
<th>SepalLength</th>
<th>SepalWidth</th>
<th>PetalLength</th>
<th>PetalWidth</th>
<th>SpeciesPred</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.7</td>
<td>2.8</td>
<td>6.7</td>
<td>2</td>
<td>virginica</td>
</tr>
<tr>
<td>5.2</td>
<td>3.4</td>
<td>1.4</td>
<td>0.2</td>
<td>setosa</td>
</tr>
<tr>
<td>6.1</td>
<td>3</td>
<td>4.9</td>
<td>1.8</td>
<td>virginica</td>
</tr>
<tr>
<td>5</td>
<td>3.5</td>
<td>1.3</td>
<td>0.3</td>
<td>setosa</td>
</tr>
<tr>
<td>5.4</td>
<td>3.9</td>
<td>1.7</td>
<td>0.4</td>
<td>setosa</td>
</tr>
<tr>
<td>4.9</td>
<td>2.5</td>
<td>4.5</td>
<td>1.7</td>
<td>virginica</td>
</tr>
<tr>
<td>5.1</td>
<td>3.5</td>
<td>1.4</td>
<td>0.3</td>
<td>setosa</td>
</tr>
<tr>
<td>5.8</td>
<td>2.6</td>
<td>4</td>
<td>1.2</td>
<td>versicolor</td>
</tr>
<tr>
<td>5.9</td>
<td>3</td>
<td>4.2</td>
<td>1.5</td>
<td>versicolor</td>
</tr>
<tr>
<td>6.9</td>
<td>3.1</td>
<td>5.4</td>
<td>2.1</td>
<td>virginica</td>
</tr>
</tbody>
</table>

Here, the accuracy of *SpeciesPred* values would ultimately depend on the strength of the child model used to score, indicated by the model’s Evaluation Metrics.
Regression

The Regression node performs “supervised” learning algorithms to find a best fit relationship between independent and dependent variables in a data set. Once a relationship is found, new data can be compared to the Regression model to make predictions.

Regression works best when attempting to output continuous values.
For example, you could apply a Regression node to a data set containing information about product pricing and order quantities to determine how pricing affects demand.
Once the Regression node had determined the relationship between the input variables, you could use that relationship to make predictions.

Regression: Training

To perform Regression training, you will need a labeled data set - that is, one where the desired output value is already known. This data set also needs to contain a set of fields, or parameters, that the Analytic Model can associate with values in the Label field.

Loan Example: Training

As a simple example of Regression training, consider a sample from a Loan data set. Suppose that the object of training is to suggest interest rates for applicants based on parameterized information about the applicant. (See Loan Data Set 1).
Field names: amt = Loan Amount; yEm = Years Employed; dti = Debt to Income Ratio; inc = Annual Income; intR = Interest Rate.
Using the Analysis Designer

Analysis Designer Nodes

Loan Data Set 1

<table>
<thead>
<tr>
<th>id</th>
<th>amt</th>
<th>yEm</th>
<th>dti</th>
<th>inc</th>
<th>intR</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>14000</td>
<td>10</td>
<td>27</td>
<td>68000</td>
<td>12.29</td>
</tr>
<tr>
<td>002</td>
<td>25000</td>
<td>1</td>
<td>20.09</td>
<td>85000</td>
<td>14.65</td>
</tr>
<tr>
<td>003</td>
<td>6000</td>
<td>10</td>
<td>27.8</td>
<td>85000</td>
<td>12.69</td>
</tr>
<tr>
<td>004</td>
<td>15600</td>
<td>1</td>
<td>13.37</td>
<td>95000</td>
<td>9.17</td>
</tr>
<tr>
<td>005</td>
<td>9250</td>
<td>1</td>
<td>21.76</td>
<td>70000</td>
<td>9.99</td>
</tr>
<tr>
<td>006</td>
<td>2500</td>
<td>1</td>
<td>14.8</td>
<td>45000</td>
<td>9.99</td>
</tr>
<tr>
<td>007</td>
<td>10000</td>
<td>5</td>
<td>14.97</td>
<td>93000</td>
<td>9.17</td>
</tr>
<tr>
<td>008</td>
<td>20000</td>
<td>1</td>
<td>11.81</td>
<td>135000</td>
<td>14.65</td>
</tr>
<tr>
<td>009</td>
<td>3600</td>
<td>1</td>
<td>29.69</td>
<td>29999</td>
<td>15.61</td>
</tr>
<tr>
<td>010</td>
<td>20150</td>
<td>10</td>
<td>27.55</td>
<td>48000</td>
<td>17.86</td>
</tr>
</tbody>
</table>

For training, 4 parameters could be used as Input Fields: amount (amt), yEm, dti, and income (inc). The interest rate (intR) could be used as the Label Field.

After specifying Inputs and the Label field, you would then create a Prediction Field with an appropriate name and the same data type as your Label Field. In this case, a field named *Suggested Interest Rate* (sIR) of the Decimal data type will work.

After running your Analysis, the Regression node would output a data set containing all specified Inputs, the Label Field, the Prediction Field, and any other fields associated to records. (See Loan Data Set 2).

Importantly, training will also create a child model within the selected Analytic Model. This child model could later be used for scoring.

Loan Data Set 2

<table>
<thead>
<tr>
<th>id</th>
<th>amt</th>
<th>yEm</th>
<th>dti</th>
<th>inc</th>
<th>intR</th>
<th>sIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>14000</td>
<td>10</td>
<td>27</td>
<td>68000</td>
<td>12.29</td>
<td>12.68</td>
</tr>
<tr>
<td>002</td>
<td>25000</td>
<td>1</td>
<td>20.09</td>
<td>85000</td>
<td>14.65</td>
<td>12.23</td>
</tr>
<tr>
<td>003</td>
<td>6000</td>
<td>10</td>
<td>27.8</td>
<td>85000</td>
<td>12.69</td>
<td>12.22</td>
</tr>
<tr>
<td>004</td>
<td>15600</td>
<td>1</td>
<td>13.37</td>
<td>95000</td>
<td>9.17</td>
<td>10.06</td>
</tr>
<tr>
<td>005</td>
<td>9250</td>
<td>1</td>
<td>21.76</td>
<td>70000</td>
<td>9.99</td>
<td>12.17</td>
</tr>
<tr>
<td>006</td>
<td>2500</td>
<td>1</td>
<td>14.8</td>
<td>45000</td>
<td>9.99</td>
<td>10.65</td>
</tr>
<tr>
<td>007</td>
<td>10000</td>
<td>5</td>
<td>14.97</td>
<td>93000</td>
<td>9.17</td>
<td>10.08</td>
</tr>
</tbody>
</table>
Using the Analysis Designer

Regression: Evaluation

To evaluate the accuracy of an Analytic Model - and by extension the accuracy of scoring that is performed using that model - you can use the Regression node’s Evaluate operation. To evaluate a child training model, you need to use a Validation dataset as an input to a Regression node.

Loan Example: Evaluating the Child Training Model

To evaluate the child model created during the Loan data set training, you would need to input a Validation data set into a Regression node and select the Evaluate operation. The Evaluation would then produce an RMSE.

<table>
<thead>
<tr>
<th>ModelDisplayName</th>
<th>ChildModelDisplayName</th>
<th>Rank</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression Model</td>
<td>Child Model 1</td>
<td>1</td>
<td>3.82</td>
</tr>
</tbody>
</table>

Regression: Re-Training, Re-Evaluating

After Training and Evaluating your first child model, you can always choose to Train another one, in order to obtain a better RMSE and more accurate scoring results. To retrain, you can use new data and/or different parameters as input fields. Each time you Re-Train using the same Analytic Model, another “Child Model” will be produced.

Loan Example: Re-Training and Re-Evaluating

Here as an example, suppose that we Re-Trained the Loan data set’s Analytic Model with 2 additional parameters for each record: open_acc and msld. Where: open_acc = Number of credit lines open in lendee’s file; and, msld = Months since last delinquency.

To Re-Train, we could edit the original Analysis by adding these parameters as Input Fields in the Regression node and rebuilding the Analysis. Upon rebuild, the Analysis would push a new child model to the original Analytic Model. The Analysis would also output a new data set, containing all 6 Input Fields, the Label field, and the Prediction field. (This Data Set is not listed).
To determine the effect of adding 2 parameters to training, we would then use the Regression node’s Evaluate operation as before. This time, though, we would select the new child model.

In this case, suppose that doing so produces a slightly better RMSE:

<table>
<thead>
<tr>
<th>ModelDisplayName</th>
<th>ChildModelDisplayName</th>
<th>Rank</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression Model</td>
<td>Child Model 2</td>
<td>1</td>
<td>3.79</td>
</tr>
</tbody>
</table>

**Regression: Scoring**

Once you have selected a child model within your Analytic Model to use for scoring, you can create another Analysis that uses a Regression node to score an unlabeled data set - that is, to predict values for each record.

To do so, you will need a data set containing the same fields that were used as parameters when the scoring model was trained. During scoring, the Regression node will compare the values in these fields to values in the scoring model.

**Loan Example: Scoring**

To complete our Loan data set example, we would first need to select a child model to use for scoring. In our example, Child Model 2 performed slightly better - so, this is the model we would choose. In practice, however, we could continue to retrain and re-evaluate with new data and/or different parameters to achieve an even better model.

To properly score using Child Model 2, we would need an unlabeled data set containing the model’s six Input fields. (See Loan Data Set 3).

**Loan Data Set 3**

<table>
<thead>
<tr>
<th>id</th>
<th>amt</th>
<th>yEm</th>
<th>dti</th>
<th>inc</th>
<th>open_acc</th>
<th>msld</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>6000</td>
<td>2</td>
<td>2.98</td>
<td>50000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>002</td>
<td>35000</td>
<td>10</td>
<td>14.39</td>
<td>86000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>003</td>
<td>10000</td>
<td>1</td>
<td>24.44</td>
<td>60000</td>
<td></td>
<td>59</td>
</tr>
<tr>
<td>004</td>
<td>25675</td>
<td>10</td>
<td>18.8</td>
<td>95000</td>
<td></td>
<td>21</td>
</tr>
<tr>
<td>005</td>
<td>20000</td>
<td>2</td>
<td>17.18</td>
<td>200000</td>
<td></td>
<td>31</td>
</tr>
<tr>
<td>006</td>
<td>9900</td>
<td>1</td>
<td>21.96</td>
<td>45000</td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>007</td>
<td>10000</td>
<td>10</td>
<td>10.22</td>
<td>150000</td>
<td></td>
<td>11</td>
</tr>
</tbody>
</table>
We would then need to input this data set into a Regression node and select the Score operation. Doing so might produce the following results. (See Loan Data Set 4).

**Loan Data Set 4**

<table>
<thead>
<tr>
<th>id</th>
<th>amt</th>
<th>yEm</th>
<th>dti</th>
<th>inc</th>
<th>open_acc</th>
<th>msld</th>
<th>sIR</th>
</tr>
</thead>
<tbody>
<tr>
<td>008</td>
<td>14000</td>
<td>1</td>
<td>12.39</td>
<td>110000</td>
<td>11</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>009</td>
<td>18000</td>
<td>7</td>
<td>36.91</td>
<td>85000</td>
<td>15</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>010</td>
<td>28000</td>
<td>6</td>
<td>18.09</td>
<td>165000</td>
<td>17</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This data set could then be output to a new data store and used in other Data Stages, such as a Dashboard.

**Recommendation**

The Recommendation node can be used to find new items a customer might be interested in.

To do so, the Recommendation node requires a data set containing at least 3 fields: a field that uniquely identifies Users, a field that uniquely identifies Products, and, most importantly, a numeric Ranking field that represents how a given User has rated a Product.
With these 3 fields, the Recommendation node can compare the rankings that different users have given to the same products to predict how users might rank products they haven’t experienced yet.

For example, given the scenario in the diagram above, the Recommendation node could predict how User 002 might rank Product C. This Prediction would be based off of how User 001 ranked Product C, given that User 001 and User 002 ranked Products A and B quite similarly. In cases where predicted rankings are high, the new product could then be recommended to the user.

**Recommendation Training**

To perform Recommendation training, you will need a labeled data set - that is, one where users have ranked products.

**Product Recommendation Example: Training**

As a simple example of Recommendation training, consider a sample from a Product rankings data set. Suppose that the object of training is to predict how users might rank products they haven’t used. (See Product Ranking Data Set 1).
Using the Analysis Designer

Analysis Designer Nodes

Product Ranking Data Set 1

<table>
<thead>
<tr>
<th>user</th>
<th>product</th>
<th>ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>a</td>
<td>99</td>
</tr>
<tr>
<td>001</td>
<td>b</td>
<td>42</td>
</tr>
<tr>
<td>001</td>
<td>c</td>
<td>73</td>
</tr>
<tr>
<td>002</td>
<td>a</td>
<td>33</td>
</tr>
<tr>
<td>002</td>
<td>b</td>
<td>63</td>
</tr>
</tbody>
</table>

For training, *ranking* would be specified as the Rating field, *user* would be specified as the User field, and *product* would be specified as the Product field.

After specifying these inputs, you would then create a Prediction Field with an appropriate name. In this case, a field named *predictedRank* will work.

After running the Training Analysis, the Recommendation node will output a data set containing the User and Product input fields, along with the Prediction Field in place of the Rating field. (See Product Ranking Data Set 2).

Product Ranking Data Set 2

<table>
<thead>
<tr>
<th>user</th>
<th>product</th>
<th>predictedRank</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>a</td>
<td>99</td>
</tr>
<tr>
<td>001</td>
<td>b</td>
<td>35</td>
</tr>
<tr>
<td>001</td>
<td>c</td>
<td>68</td>
</tr>
<tr>
<td>002</td>
<td>a</td>
<td>40</td>
</tr>
<tr>
<td>002</td>
<td>b</td>
<td>51</td>
</tr>
</tbody>
</table>

Importantly, training will also create a child model within the selected Analytic Model. This child model could later be used for scoring.

Recommendation: Evaluation

To evaluate the accuracy of an Analytic Model - and by extension the accuracy of scoring that is performed using that model - you can use the Recommendation node’s Evaluate operation.

To evaluate a child training model, you need to use a Validation dataset as an input to a Recommendation node.
Product Recommendation Example: Evaluating the Child Training Model

To evaluate the child model created during the Product Ranking data set training, you would need to input a Validation data set into a Recommendation node and select the Evaluate operation.

The Evaluation would then produce an RMSE.

<table>
<thead>
<tr>
<th>ModelDisplayName</th>
<th>ChildModelDisplayName</th>
<th>Rank</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recommendation</td>
<td>Child Model 1</td>
<td>1</td>
<td>15.82</td>
</tr>
</tbody>
</table>

Recommendation: Re-Training, Re-Evaluating

After Training and Evaluating your first child model, you can always choose to Train another one, in order to obtain a better RMSE and more accurate scoring results.

To retrain for Recommendation, you’ll need new data. Each time you Re-Train using the same Analytic Model, another “Child Model” will be produced.

Once a new child model is produced, you can then evaluate it using the Data Store Output and child model that was produced by your new training attempt. If a new child model is found to have a lower RMSE, you could then use it for scoring.

Recommendation: Scoring

Once you have selected a child model within your Analytic Model to use for scoring, you can create another Analysis that uses a Recommendation node to score an unlabeled data set - that is, to predict values for each record.

Recommendation currently features 3 types of Scoring:

- **Ratings**
  
  Given a User field and a Product field, predict a Rating field that represents how that User might rate that Product.

- **Users**
  
  Given a Product field, find Ratings that were given to the Product.

- **Products**
  
  Given a User field, find Products to recommend to the User.

Product Recommendation Example: Scoring

To complete our Product Recommendation data set example, we would first need to select a child model to use for scoring. In our example, we only trained one model, Child Model 1 - so, this is the model we would choose. In practice, however, we could continue to retrain and re-evaluate to achieve an even better model.
To derive predictions for Product Rankings using Child Model, we would need an unlabeled data set containing a user field and a product field. Additionally, the set of values in these fields would need to be the same as the set of values used in Training (see Product Ranking Data Set 1).

**Product Ranking Data Set 3**

<table>
<thead>
<tr>
<th>user</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>a</td>
</tr>
<tr>
<td>001</td>
<td>b</td>
</tr>
<tr>
<td>001</td>
<td>c</td>
</tr>
<tr>
<td>002</td>
<td>a</td>
</tr>
<tr>
<td>002</td>
<td>b</td>
</tr>
<tr>
<td>002</td>
<td>c</td>
</tr>
</tbody>
</table>

We would then need to input this data set into a Recommendation node and select the Score operation. In addition, we’d need to select: Ratings as our Score Type, user as our User field, and product as our Product field. Like with Training, we’d also need to create a Prediction field to hold predicted Rating values. In this case, we can again use the name predictedRank and specify the Integer data type. Doing so might produce the following results. (See Product Ranking Set 4).

**Product Ranking Data Set 4**

<table>
<thead>
<tr>
<th>user</th>
<th>product</th>
<th>predictedRank</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>a</td>
<td>91</td>
</tr>
<tr>
<td>001</td>
<td>b</td>
<td>30</td>
</tr>
<tr>
<td>001</td>
<td>c</td>
<td>76</td>
</tr>
<tr>
<td>002</td>
<td>a</td>
<td>46</td>
</tr>
<tr>
<td>002</td>
<td>b</td>
<td>50</td>
</tr>
<tr>
<td>002</td>
<td>c</td>
<td>60</td>
</tr>
</tbody>
</table>

This data set could then be output to a new data store and used in other Data Stages, such as a Dashboard. Note that in this example, the Scoring model was able to generate a prediction about how user 002 would rank Product C, based on how user 001 ranked products A, B, and C.
Forecast

The Forecast node allows you to use time series data to attempt to predict the future values of a measure. To do so, Forecast requires a field containing a numeric measure. For best results, this Field should also be sorted by date before it is used in Training. The end result, or output, of a Forecast is ultimately a data set containing projected values for a specific measure, for future time periods.

**Forecast: Training**

To perform Forecast training, you will need a dataset that contains a measure for which you would like to make predictions. For best results, this data set should also contain a Date or DateTime field.

**Forecast Example: Training**

As a simple example of Forecast training, consider a sample from a Sales data set. Suppose that the object of training is to make a model for sales projections, based on historical sales amounts. (See Sales Data Set 1).

Field names: date = Sales Date; amt = Sales Amount

**Sales Data Set 1**

<table>
<thead>
<tr>
<th>date</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-04-02</td>
<td>14333.75</td>
</tr>
<tr>
<td>2016-06-18</td>
<td>16216.27</td>
</tr>
<tr>
<td>2016-02-19</td>
<td>18706.21</td>
</tr>
<tr>
<td>2016-03-12</td>
<td>15331.75</td>
</tr>
<tr>
<td>2016-07-16</td>
<td>13550.19</td>
</tr>
<tr>
<td>2016-02-05</td>
<td>24924.5</td>
</tr>
</tbody>
</table>

**Sorting the Training Set by DateTime**

Since the Forecast node expects the selected Input Field, i.e. the field you want to predict on, to be a set of measures over time, it needs to be sorted before training occurs.

If the dataset is not already sorted, you can use the Forecast node’s Other tab, which contains a Sort option.
Using the Analysis Designer

Nodes

Sorted Sales Data Set 1

<table>
<thead>
<tr>
<th>date</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016-02-05</td>
<td>24924.5</td>
</tr>
<tr>
<td>2016-02-19</td>
<td>18706.21</td>
</tr>
<tr>
<td>2016-03-12</td>
<td>15331.75</td>
</tr>
<tr>
<td>2016-04-02</td>
<td>14333.75</td>
</tr>
<tr>
<td>2016-06-18</td>
<td>16216.27</td>
</tr>
<tr>
<td>2016-07-16</td>
<td>13550.19</td>
</tr>
</tbody>
</table>

Model Tab

Once the data set is sorted, it can then be input into the Forecast node. Within the Model tab, we would select `amt` as our Input Field and Train as our Operation. As with all other Analytics nodes, we will also need to select an Analytic Model in which to store the child model created by Training. In this case, assume that we have created a Forecasting Analytic Model beforehand.

Forecast Tab

After making specifications in the Model tab, we can then move on to the Forecast Tab. For this simple example, we will select 5 Future Periods to Forecast, and use the Auto Fit setting. At this point, users more familiar with Forecasting and ARIMA models can also use the Forecast Tab to make more granular modifications to their model.

Output of Training

After running a Forecasting Analysis, the Forecast node will output a data set containing our original Input Field (`amt`) and three new fields: `TimePeriod`, `HistoricalValue`, and `IsForecast` (See Sales Data Set 2).

Importantly, Training will also create a child model within the selected Analytic Model. This child model could later be used for scoring.

Sales Data Set 2 - Output of Forecast Training

<table>
<thead>
<tr>
<th>TimePeriod</th>
<th>HistoricalValue</th>
<th>IsForecast</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>24924.5</td>
<td>False</td>
<td>13705.16</td>
</tr>
<tr>
<td>1</td>
<td>18706.21</td>
<td>False</td>
<td>13674.80</td>
</tr>
<tr>
<td>3</td>
<td>15331.75</td>
<td>False</td>
<td>13924.12</td>
</tr>
<tr>
<td>4</td>
<td>14333.75</td>
<td>False</td>
<td>14434.62</td>
</tr>
</tbody>
</table>
6 ■ Using the Analysis Designer

Analysis Designer Nodes

<table>
<thead>
<tr>
<th>TimePeriod</th>
<th>HistoricalValue</th>
<th>IsForecast</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>16216.27</td>
<td>False</td>
<td>14572.03</td>
</tr>
<tr>
<td>6</td>
<td>13550.19</td>
<td>False</td>
<td>13324.24</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>True</td>
<td>14184.63</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>True</td>
<td>14134.72</td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>True</td>
<td>14135.31</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>True</td>
<td>14135.28</td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>True</td>
<td>14135.28</td>
</tr>
</tbody>
</table>

**Fields in Forecast Training Output**

Note that these fields can be output to a Data Store for usage in other Data Stages, but that they don’t need to be.

*TimePeriod*

The Forecast node creates a Time Period for each record in the Input Field supplied. It will also create an additional number of Time Periods equivalent to the Number of Future Periods Forecast.

*HistoricalValue*

This field holds the actual value of the Input Field during the associated Time Period. In this example, note that *HistoricalValue* values match the values in the *amt* field in Sales Data Set 1. Also note that Forecast records do not have Historical Values.

*IsForecast*

True for records that represent forecasts; False for records that represent Historical Values.

*Selected Input Field*

These are the values the Forecast model has decided to assign to your selected Input Field. In cases where *IsForecast* is False, the values represent a best fit of the *HistoricalValue* at that time. In cases where *IsForecast* is True, the values are your projections.

*Forecast: Evaluate and Re-Train*

Once you have Trained a Forecast child model, you can decide if you’d like to Score with the model or if you’d like to try to Evaluate the model and create other models with different parameters. With Forecast, there is no explicit Evaluate operation, however you can use the output of Training to compare Historical Values to Forecast Values in a new Analysis or a Dashboard.
After doing so, you can then decide whether you’d like to try to create a different child model by making different specifications in the Forecast node’s properties panel and Training again.

**Forecast: Scoring**

Once you have a child model you are satisfied with - which you’ve specified as the Scoring Model within your Analytic Model - you can Forecast on new data. To do so, simply create an Analysis that uses the Forecast node’s Score operation.

The process of Scoring is nearly identical to Training:

- Use a Sort node to sort your data by date if it is not already sorted.
- Select your Analytic Model. The specified scoring model will be used to make predictions.
- Select an Input Field to forecast on. This should be the same type of field that was used during Training.
- Specify Number of Future Periods to Forecast - like Training, these will be your predictions.

After you have Run your Forecast Scoring Analysis, your output will be formatted just like it was in the Training output, with *TimePeriod*, *HistoricalValue*, *IsForecast*, and your Input Field. As with Training, the values in your Input Field where *IsForecast* is True are your predictions.

**Text Analytics nodes**

The Text Analytics nodes allow you to analyze text in an input string field.

See:

- Identify Key Phrases
- Sentiment Analysis
- Detect Entities
- Detect Entities in Medical Text

**Identify Key Phrases**

Note: This node is only available with the Cloud edition of the product.

The Identify Key Phrases node allows you to identify specific words and phrases in an input string field. A 'key phrase' is a string containing a noun; this could be a single word, or a modifier and a noun. The node uses the Amazon Comprehend API to identify key phrases in incoming text fields.
Using the Analysis Designer

Analysis Designer Nodes

The node outputs all input fields, and one new array field called 'IdentifiedPhrases'. You can click a row in the IdentifiedPhrases column to show the following information:

- BeginOffset - The position of the first character of the phrase in the string.
- EndOffset - The position of the last character of the phrase in the string.
- Score - The estimated accuracy of the analysis.
- Text - The detected text entity.

Example
Input text: *It was a big house, with large windows in the countryside.*

<table>
<thead>
<tr>
<th>BeginOffset</th>
<th>EndOffset</th>
<th>Score</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>17</td>
<td>0.9974726</td>
<td>big house</td>
</tr>
<tr>
<td>25</td>
<td>37</td>
<td>0.99040056</td>
<td>large windows</td>
</tr>
<tr>
<td>46</td>
<td>56</td>
<td>0.9912345</td>
<td>countryside</td>
</tr>
</tbody>
</table>

For more information, please see the Amazon Comprehend documentation, for example: https://docs.aws.amazon.com/comprehend/latest/dg/how-key-phrases.html

Properties
Display Name
Specify the name of the node that is displayed on the Analysis Designer canvas. The default value is Identify Key Phrases.

Identify Key Phrases in
Select an input field to analyze.

Language
Select the language of the text to analyze. The default value is English.

Sentiment Analysis
Note: This node is only available with the Cloud edition of the product.

The Sentiment Analysis node allows you to inspect the text in an input string field to gauge the sentiment. For example, you can use this node to infer the sentiment of comments or call center notes. The node uses the Amazon Comprehend API to determine the sentiment of the text in incoming string fields. The possible sentiment values are:

- Positive
Using the Analysis Designer

Analysis Designer Nodes

- Negative
- Neutral
- Mixed

The node outputs all input fields, and the following new fields:
- Sentiment - The inferred sentiment of the string.
- SentimentScoreMixed - The confidence level that the sentiment of the string is mixed.
- SentimentScorePositive - The confidence level that the sentiment of the string is positive.
- SentimentScoreNegative - The confidence level that the sentiment of the string is negative.
- SentimentScoreNeutral - The confidence level that the sentiment of the string is neutral.

Example
Input text: Today I won a prize, I am so happy.

<table>
<thead>
<tr>
<th>Sentiment</th>
<th>SentimentScoreMixed</th>
<th>SentimentScorePositive</th>
<th>SentimentScoreNegative</th>
<th>SentimentScoreNeutral</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITIVE</td>
<td>0.00335512369668</td>
<td>0.986912340789</td>
<td>0.0012341230975</td>
<td>0.008563112124</td>
</tr>
<tr>
<td></td>
<td>968547</td>
<td>7323</td>
<td>521897</td>
<td>698635</td>
</tr>
</tbody>
</table>

For more information, please see the Amazon Comprehend documentation, for example https://docs.aws.amazon.com/comprehend/latest/dg/how-key-phrases.html

Properties
Display Name
Specify the name of the node that is displayed on the Analysis Designer canvas. The default value is Sentiment Analysis.

Identify Key Phrases in
Select an input field to analyze.

Language
Select the language of the text to analyze. The default value is English.
Detect Entities

Note: This node is only available with the Cloud edition of the product.

The Detect Entities node allows you to identify text entities in an input string field. An entity is a textual reference to the unique name of an object, such as people, places, dates and quantities. The node uses the Amazon Comprehend API to identify entities in incoming text fields.

The node outputs all input fields, and one new array field called 'DetectedEntities'. You can click a row in the DetectedEntities column to show the following information:

- **BeginOffset** - The position of the first character of the word in the string.
- **EndOffset** - The position of the last character of the word in the string.
- **Score** - The estimated accuracy of the analysis.
- **Text** - The detected text entity.
- **Type** - The entity type. The following table lists the entity types which are recognized by the node:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMERCIAL_ITEM</td>
<td>A branded product.</td>
</tr>
<tr>
<td>DATE</td>
<td>A date (e.g. 12/22/2018), a day (e.g. Thursday), a month (e.g. May) or a time (e.g. 9:15 a.m.).</td>
</tr>
<tr>
<td>EVENT</td>
<td>An event for example a concert or an election.</td>
</tr>
<tr>
<td>LOCATION</td>
<td>A location such as a country, a city, a river or a building.</td>
</tr>
<tr>
<td>ORGANIZATION</td>
<td>An organization such as a company, a sports team or a religion.</td>
</tr>
<tr>
<td>OTHER</td>
<td>Entities that do not fit in any of the other categories.</td>
</tr>
<tr>
<td>PERSON</td>
<td>Individuals, groups of people, nicknames, fictional characters.</td>
</tr>
<tr>
<td>QUANTITY</td>
<td>A quantified amount, such as currency, percentages, numbers or bytes.</td>
</tr>
<tr>
<td>TITLE</td>
<td>An official name given to any creation or creative work, such as movies, books, or songs.</td>
</tr>
</tbody>
</table>

For more information, please see the Amazon Comprehend documentation, for example https://docs.aws.amazon.com/comprehend/latest/dg/how-entities.html
Using the Analysis Designer

Analysis Designer Nodes

Example
Input text: The customer is John Smith, a 50 year old who lives in New York.
In this example, the following entities can be identified:

<table>
<thead>
<tr>
<th>BeginOffset</th>
<th>EndOffset</th>
<th>Score</th>
<th>Text</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>17</td>
<td>26</td>
<td>0.9914761</td>
<td>John Smith</td>
<td>PERSON</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
<td>0.9981457</td>
<td>50</td>
<td>QUANTITY</td>
</tr>
<tr>
<td>56</td>
<td>63</td>
<td>0.8912875</td>
<td>New York</td>
<td>LOCATION</td>
</tr>
</tbody>
</table>

Properties
Display Name
Specify the name of the node that is displayed on the Analysis Designer canvas. The default value is Detect Entities.

Detect Entities
Select an input string field to analyze.

Language
Select the language of the text to analyze. The default value is English.

Detect Entities in Medical Text
Note: This node is only available with the Cloud edition of the product.

The Detect Entities in Medical Text node allows you to inspect an input string field to identify words that are medical entities. Medical entities can include Protected Health Information (PHI) and medical conditions, for example. The node uses the Amazon Comprehend API to identify medical entities in incoming text fields.

The node outputs all input fields, and one new array field called 'DetectedEntities'. You can click a row in the DetectedEntities column to show the following information:

- BeginOffset - The position of the first character of the entity in the string.
- EndOffset - The position of the last character of the entity in the string.
- Score - The estimated accuracy of the analysis.
- Text - The detected text entity.
- Type - The entity type, for example 'AGE' or 'NAME'.
- Category - The entity category, for example 'PROTECTED_HEALTH_INFORMATION'.

Using the Analysis Designer

**Analysis Designer Nodes**

- **Id** - A record ID.
- **Response** - The complete API response in JSON format. This response can be processed further for traits and attribute information.

**Example**

Input text: *The patient is a 40 year old teacher.*

<table>
<thead>
<tr>
<th>BeginOffset</th>
<th>EndOffset</th>
<th>Score</th>
<th>Text</th>
<th>Type</th>
<th>Category</th>
<th>Id</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>19</td>
<td>0.9981877</td>
<td>40</td>
<td>AGE</td>
<td>PROTECTED_HEALTH_INFORMATION</td>
<td>0</td>
<td>{ &quot;Id&quot;:0, &quot;beginOffset&quot;:18, &quot;endOffset&quot;:19, &quot;score&quot;:0.9981877, &quot;text&quot;:&quot;40&quot;, &quot;traits&quot;:[], &quot;attributes&quot;:[], &quot;categoryAsString&quot;:&quot;PROTECTED_HEALTH_INFORMATION&quot;, &quot;typeAsString&quot;:&quot;AGE&quot;}</td>
</tr>
</tbody>
</table>
Using the Analysis Designer

Analysis Designer Nodes

<table>
<thead>
<tr>
<th>BeginOffset</th>
<th>EndOffset</th>
<th>Score</th>
<th>Text</th>
<th>Type</th>
<th>Category</th>
<th>Id</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>36</td>
<td>0.536245</td>
<td>teacher</td>
<td>PROFESSION</td>
<td>PROTECTED_HEALTH_INFORMATION</td>
<td>1</td>
<td>{ &quot;Id&quot;:0, &quot;beginOffset&quot;:30, &quot;endOffset&quot;:36, &quot;score&quot;:0.53624517, &quot;text&quot;:&quot;teacher&quot;, &quot;traits&quot;:[],&quot;attributes&quot;:[], &quot;categoryAsString&quot;: &quot;PROTECTED_HEALTH_INFORMATION&quot;, &quot;typeAsString&quot;: &quot;PROFESSION&quot; }</td>
</tr>
</tbody>
</table>

For more information, please see the Amazon Comprehend documentation, for example https://docs.aws.amazon.com/comprehend/latest/dg/how-medical-phi.html

**Properties**

**Display Name**
Specify the name of the node that is displayed on the Analysis Designer canvas. The default value is Detect Entities in Medical Text.

**Detect Medical Entities in**
Select an input field to analyze.

**Detect PHI Only**
Select this option to return only entities that are categorized as PHI (Protected Health Information).
System nodes

Repartition
The Repartition node can be used to control how data is partitioned during Analysis execution.

Partition By Fields
This parameter allows you to choose which fields to use when repartitioning the data set.

Number of Partitions
This parameter allows you to choose how many partitions the data set should be divided into.

Repartitioning Example
Suppose you had the following data set.

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
</tr>
<tr>
<td>D</td>
<td>13</td>
</tr>
<tr>
<td>A</td>
<td>14</td>
</tr>
<tr>
<td>A</td>
<td>15</td>
</tr>
<tr>
<td>C</td>
<td>16</td>
</tr>
<tr>
<td>B</td>
<td>17</td>
</tr>
<tr>
<td>C</td>
<td>18</td>
</tr>
<tr>
<td>B</td>
<td>19</td>
</tr>
</tbody>
</table>

Were you to select name as a Partition By Field and specify 4 as the Number of Partitions, the Repartition node might produce the following result.

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>18</td>
</tr>
<tr>
<td>C</td>
<td>12</td>
</tr>
<tr>
<td>C</td>
<td>16</td>
</tr>
</tbody>
</table>
Within your result data set, records with similar Partition By Field values are placed within the same partition - that is, within close proximity of one another within the data set - in no particular order. Additionally, the specified Number of Partitions parameter matches the number of unique values within the Partition By Field.

**Cache Data**

The Cache Data node can be used to save time on Analysis runs that process a large number of records, and where the global Cache Output of Nodes Analysis setting has been turned off.

Caching is most useful at points in an Analysis where the flow of execution is split. For example, consider a situation where a Data Store is sorted by two separate criteria.

<table>
<thead>
<tr>
<th>name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>A</td>
<td>14</td>
</tr>
<tr>
<td>A</td>
<td>15</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
</tr>
<tr>
<td>B</td>
<td>17</td>
</tr>
<tr>
<td>B</td>
<td>19</td>
</tr>
<tr>
<td>D</td>
<td>13</td>
</tr>
</tbody>
</table>
Here, all records in the Data Store need to be recomputed for each Sort node. With large Data Stores, this can drastically increase execution time.

With the Cache Data node, data can be cached for the split, eliminating the need to recompute. This can drastically reduce execution time, particularly for large data sets.

The Cache Output of Nodes Setting
By default, the Analysis Designer will cache the output of all nodes automatically. i.e. the “Cache Output of Nodes” check box found within Analysis Settings will be checked. Effectively, this means that Analyses will behave as if there were a Cache Data node after
every single node, by default (with the exception of Data Store Outputs). It also means that if you want to use a Cache Data node at a specific point, you should turn off the global Cache Output of Nodes setting first.

In general, caching the output of all nodes is recommended on smaller data sets, to save time. For example, caching could be used in the following Analysis, which processes 28,000 records.

With caching turned off, however, this same Analysis takes about twice as long to run. This is due to the fact that some operations have to visit the previous node and recompute the incoming data set - which takes time. In the following diagram, this is visualized in places where the number of records processed at a node is a multiple of 28,000, that is greater than 28,000 (the number of records in the data set).

Global Caching Can Reduce Performance on Large Datasets
It should be noted that with large data sets, caching all nodes can actually have a negative impact on performance, since the amount of time it takes to create the memory cache is greater than recomputation.
For example, experimentation has found that repeatedly sorting 2,000,000 records (using different sort criteria for each sort) without any caching takes approximately 63% as long as it would when global caching is turned on.

Without caching, nodes 1, 2, and 3 must be revisited multiple times, however at 2,000,000 input records this is still faster than global caching.

**Showing and Working with Sheets**

When building new Analyses, it is recommended that you keep the Show Sheets feature toggled on.

Every time you add a new node to your Analysis, a new sheet will be generated automatically. Each sheet will then display what is happening to your fields at that point in the Analysis.

**Adding New Columns**

When working with Sheets, you can add new columns (i.e., Fields) that are comprised of functions that manipulate other Fields.

New Columns are added using the New Column button. Once columns are added, they can be treated like any other Field and pushed to new Analysis Designer nodes.
Testing an Analysis

While building your Analysis, you can run test executions from within the Analysis Designer. Testing an Analysis will manipulate data fields as specified in each designer node. These manipulations will appear in each node’s sheet to illustrate what will happen each time the Analysis is executed.

Additional Analysis Settings

Log Limit
This setting controls the maximum number of errors that will be displayed in the Analysis Designer’s Output Log when Testing. It also acts as a maximum threshold for the number of errors you want the executor to allow before exiting the Analysis run and marking the run as Failed.

Sampling Data for Testing
This setting controls the sample size of data used for Testing an Analysis. Note that this setting will interact with the Sampling settings made in Data Store Inputs and Sample nodes. Namely, settings made here are applied before settings made at the node level.

Cache Output of Nodes
This setting determines whether caching will occur for all nodes in your Analysis. Caching the output of a node can increase the speed of an Analysis when recomputes are required at points where the Analysis splits.

Caching the output of all nodes is recommended on smaller data sets, to save time. With larger data sets, however, global caching can cause a significant decrease in performance. Cache Output of Nodes is turned on by default.

Note that for larger data sets, you can use the Cache Data node to only cache the output of specific nodes. The Cache Data node is described in more detail above.

Collect Accurate Record Counts
When checked, this check box will cause the system’s Execution History to track record counts at each node within an Analysis. When unchecked, the system will not track record counts.

By default, Collect Accurate Record Counts is checked and disabled. To enable the check box, you will need to disable the Cache Output of Nodes check box. Note that Collect Accurate Record Counts must be checked when Cache Output of Nodes is checked and that the UI will enforce this.

Runtime Properties
Runtime Properties can be created by selecting a Data Store and mapping a field containing names to a field containing values. The property can then be referenced via the name field throughout the Analysis, using the RUNTIME() function.
As an example, consider the following sample data set:

**Runtime Data Set**

<table>
<thead>
<tr>
<th>id</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>100</td>
</tr>
<tr>
<td>002</td>
<td>200</td>
</tr>
<tr>
<td>003</td>
<td>300</td>
</tr>
</tbody>
</table>

If you were to select `id` as the Property Name Field and `value` as the Property Name Value, you could use `RUNTIME(id)` in any node within your analysis to return the values for each unique name in your property.

In this case, `RUNTIME(id)` would return the following:

**Results from call to RUNTIME(id)**

<table>
<thead>
<tr>
<th>New Column</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
</tr>
<tr>
<td>200</td>
</tr>
<tr>
<td>300</td>
</tr>
</tbody>
</table>

Note that values in the Property Name Value should be unique per name. If a name has more than one value associated to it, only the first value found will be returned for all records.

Also note that to use Runtime Properties in interactive mode - that is, within the sheets of nodes while building your Analysis - you need to use the *Test* button to simulate a run of the entire Analysis (rather than using Test Sheet).

**Unmasking Secure Fields in an Analysis**

If you have permission to unmask a secure field, you may unmask records in bulk using the Unmask All button. Also note that only *secure* fields may be unmasked. Fields that are encrypted but not secure will be shown as masked in Analysis sheets; however, you cannot use the Unmask all button to unmask these values.
Executing an Analysis

When an Analysis is executed, it performs the Enhancements and Analyses specified in its design and pushes manipulated data to Data Store Outputs. These Data Store Outputs can then be used by other Data Stages.

Running an Analysis

First time Run
Running an Analysis for the very first time will push manipulated data to one or more Data Store Outputs. An Analysis must be run at least once before its Data Store Outputs can be used by other Data Stages.

Re-Run
If an Analysis’ Data Store Input receives new data from its source, you will need to Re-Run the Analysis to manipulate the newly received data.

Rebuilding an Analysis

Rebuilding an Analysis will override any Data Load Range settings on Data Store Inputs and will load all data into the Analysis’ Data Store Inputs. This option may be useful when developing new Analyses for the first time.

Execution Record Counts and Caching

If global caching is turned off, you will notice record counts at nodes that are much greater (usually, by multiples of 2 or 3) than the number of known records in your Data Store Input. This is a side effect of the recomputation that occurs when caching is disabled.

For example, you may know that a Data Store has only 2,000,000 records; however, when viewing its Executions History, you find that 4,000,000 records were processed at this Data Store’s Input node.

In this particular situation, the 4,000,000 represents the fact that the Analysis’ flow of execution needed to visit the Data Store Input node twice - once for each node the flow splits into. As a result, technically, 4,000,000 records were processed, even though the Data Store only contains 2,000,000.

Field Mapping in Data Store Outputs when Analyses Change

If an Analysis has Run and you’d like to make modifications to it that will add, remove, or rename fields from your Data Store Outputs, you will first need to 1) make modifications to the Data Store Outputs outside of the Analysis and then 2) make changes within the Analysis.
Procedure for Field Addition
Consider a situation where you create an Analysis that pushes data to 4 fields in a Data Store Output.
If you were to Run this Analysis and then later decide that you’d like to add another field to the Data Store Output, you would first need to add this field from within the Data Store - that is, by selecting the Data Store within the Pipelines context and using Edit Stage.
After adding the field to the Data Store within the Edit Data Store context, you could then create the field as a column within your Analysis and map to this new, 5th field in the Data Store Output node.
When performing such a Field addition, you will also need to make sure that the field you create in the Data Store and the field you create in the Analysis are of the same Data type.

Procedure for Field Rename
If an Analysis has already Run and you want to change the name of a field in one of its Data Store Outputs, you will first need to do so from within the Data Store - that is, by selecting the Data Store in the Pipelines context and using Edit Stage.
After the name is changed from within the Data Store, you will need to re-enter the Analysis and map to the field that has the new name.

Procedure for Field Removal
If an Analysis has already Run, a field can be removed from its Data Store Output by deleting it from within the Data Store’s Edit Stage context.
After deleting the field from within the Data Store, you can then re-enter the Analysis and clear the name of the field in the Data Store Output node to remove it from the mapping.

Field Mapping when Data Store Outputs are used by other Data Stages
If you have Run an Analysis with a Data Store Output that is being used by other Data Stages, remapping can quickly become a complicated process because you will need to make your modifications in all Data Stages that use the affected Data Store.
To aid you in this process, select the Data Store Output in the Pipelines context, and use the Show Usages feature, which is accessible via the More dropdown.

Starting over with a new Data Store Output
If you have not yet reached the point where an Analysis’ Data Store Outputs are being used by other Data Stages, it can often be much easier to just Delete the Data Store Output node within your Analysis and then push to a new Data Store Output instead. This way, you do not need to worry about re-mapping at all.

Note: Deleting a Data Store Output from an Analysis does not Delete the Data Store from your Pipeline. Therefore, after removing a Data Store Output from an Analysis, you may want to Delete it from your Pipeline as well.
Scheduling Analysis Executions

Users with Administer permissions to an Analysis can schedule its Executions. Analysis Execution scheduling is primarily dependent upon the frequency at which Data Store Inputs receive new data.

For example, if you had a Data Store that received new data everyday, you could schedule an Analysis that used that Data Store to execute on a daily basis, sometime after the new data was received.

Upon Execution, the Analysis would look to its Data Store Input and perform the manipulations specified in its design on all data present, including newly arrived data. This would in turn create a new output data set, which the Analysis would push to its Data Store Output when it finished executing.

Scheduling Analysis Executions to keep up to date with newly arrived data can be used to effectively implement automated data transformation and machine learning, in real time.
Creating Data Views

After your Infogix Data3Sixty Data Stores have been configured, you can create a Data View. Data Views can be used to define which data to pull from one or multiple Data Stores, when creating Dashboards and exploring data in the Infogix Data3Sixty Visualizer.

Once a Data View is created and configured it can be scheduled to pull in the most up-to-date data - or execute - by users with Administer permissions. Data Views can also be executed manually, by anyone.

Linking Multiple Data Stores

Before you create a Data View, you’ll need to identify which Data Stores you will use. A Data View can be created using one Data Store, or multiple Data Stores. If multiple Data Stores are to be used, you will need to link them first.

To link one Data Store to another, right click one of the stores, select Edit Stage, and then enter the Links tab.

Data View Properties

Data View Properties should be set when the Data View is created; however, as you begin to create other Data Stages you may need to make modifications to your Data Views over time.

Below, you can find a high level overview of all Data View Properties. To modify these Properties, you will need at least Write permission to a Data View.

Details

This property is used to name the data view, add data stores, and select a primary data store if multiple stores are used.

Storage Type

This setting determines how the Data View’s data is stored behind the scenes. The option chosen essentially affects how quickly Dashboards that use the Data View will render.
Creating Data Views

Data View Properties

Regular Performance
Data Views that use Regular Performance will have their data stored in traditional hard drives. As such, Dashboards that use Regular Performance Data Views perform slightly slower than Dashboards that use High Performance Data Views. For this reason, it is recommended that Regular Performance is used for Dashboards that are accessed occasionally.

For example, if you have Dashboards that drill down to a very detailed level of information that are used infrequently, you would want to set them to Regular Performance.

High Performance
Data Views that use High Performance will have their data stored in solid state hard drives. As such, Dashboards that use High Performance Data Views perform slightly faster than Dashboards that use Regular Performance Data Views. For this reason, it is recommended that High Performance is used for Dashboards that are accessed frequently.

For example, if you have Dashboards that contain summary information and are used frequently, you would want to set them to High Performance, so that they always render quickly.

Big Data
On the cloud version of the product, this setting can be used to query Data Views based on Amazon Athena. On the enterprise version of the product, this setting can be used to query Data Views based on Hive/Impala. This setting is suitable for use with large amounts of data where using Amazon RedShift on the cloud or Vertica on the enterprise could be cost prohibitive. Checking this box results in a file based Data View that trades off performance for lower cost.

Table Name Prefix
This optional parameter allows you to create a prefix of up to 30 characters that will be added to the Data View’s underlying, unique name. Doing so can allow you to search for the Data View by name, when querying the database that retains Data View information, with an external tool. Without a Table Name Prefix, a Data View’s name will instead just be a non-meaningful, alphanumeric string. Adding a Table Name Prefix can make searching for the Data View easier.

Dimensions
Dimensions serve as top level containers for categorically similar data. For example, you might create a Product dimension for product data and a Customer dimension for data related to customers.

When exploring data, Dimensions will contain Fields that can be dragged-and-dropped onto the x or y-axis. Dimensions can contain the field types outlined below.
Fields
Data view fields are built using fields from data stores. When a data view is executed, its fields will populate with the information held in the selected data store field.

In general, fields contain information describing one particular aspect of a dimension. For example, a Customer dimension might have a field called Customer Name, which will populate with all customer names when the data view is run.

If a field contains numeric type data, Infogix Data3Sixty will automatically make it available as a Measure that you can perform computations on in Dashboards and the Visualizer.

Data View Field Settings
Many of these settings work similarly to those in Data Store fields, however there some notable differences; the following section highlights these.

Note: When working with Data View fields, changes to settings may not take affect until the Data View is rebuilt.

Display Format Pattern
This optional setting determines how numbers will be displayed in the Dashboards and Visualizations that use a Data View. In Data Views, Display Format Patterns work just as they do in Data Stores. In fact, if a field’s Display Format Pattern has been configured in its Data Store, these settings will simply carry over to the Data View where the field is used.

In Data View fields, Display Format Pattern can be kept the same as the fields’ settings in its Data Store, made unique, or simply left blank.

Scale
This setting controls the number of digits to the right of the decimal point that will be displayed in Dashboards and Visualizations that use a Data View’s field. For example, if value = 1.234 and Scale is set to 2, value will be displayed as 1.23 in Dashboards and Visualizations. By default, Scale settings made in Data Store fields are passed on to Data View fields.

Data Store field Scale can be overridden in a Data View field, however it is important to note that this can cause the values in each Data Stage to be different. For example, in a Data Store, if value = 1.999 and Scale is set to 3, value would be displayed as 1.999 in the Data Store UI or Analysis Sheets. If Scale were set to 2 for the same value field in a Data View, however, value would be displayed in Dashboards/Visualizations that use that Data View as 1.99.

This functionality can have an impact on Aggregate functions.
For example, consider a situation where you had 1000 records in the *value* field, all with the same *id* field value of 1, with Scale set to 3 for the *value* field in the Data Store. If you were to use *value* in an Analysis, Group By *id*, and then take the SUM of *value*, you would get 1999. If Scale were set to 2 for the same *value* field in a Data View, however, the same type of SUM aggregation would yield 1990 - a difference of 9.

**Rounding Type**

Rounding Up and Truncating in a Data View work just like Rounding up and Truncating in Data Stores.

It is important to note, however, that when Rounding Up or Truncating values in a Data View, you will be doing this to values that may have already been Rounded or Truncated in a Data Store.

By default, a field’s Rounding Type is passed from Data Store to Data View automatically, but it can be changed if needed.

**Precision**

Precision controls the number of digits to include in an integer value.

A Data View field’s Precision must always be greater than or equal to that of the value with the highest precision in a *data set*. This can be different than the Precision of a Data Store field.

For example, imagine you have a data set where the largest value for *field1* is 1,000,000,000,000. This value has a Precision of 13; so, to properly execute the Data View, *field1*’s Precision in the Data View must be 13 or greater.

Where you can run into issues is if Precision for *field1* has been set to less than 13 in its Data Store - which may lead you to believe that Precision for the *data set* is less than 13. Doing so will not cause any issues in the Data Store or Analyses, but it will cause issues in the Data View execution. Namely, a field with too low of a Precision will throw the following Error in the Executions Log:

*Unable to copy data into data store; nested exception is java.sql.SQLException: [Amazon](500310)[Amazon](500310)*

If you encounter this error, try raising Precision in applicable fields and Rebuilding the Data View. Since Data Store field Precision may actually be lower than true data set precision, you may not be able to use this as a reference. If in doubt, there is no harm in setting Precision to something very high and seeing if this will solve the issue.
7 ■ Creating Data Views

Data View Properties

Currency Code
Currency Code must be set to display currency signs in Dashboards and Visualizations that use a Data View.

You can apply Currency Codes to both values that do or don’t have Currency Format Patterning in the Data Store they come from. Note however that if the field values have a currency format, this will need to be specified in the Data Store’s Format Pattern to be properly parsed.

Location category
If a String field contains values that represent locations, you can choose a category to specify whether values represent Countries, Regions, State/Provinces, or Counties. Doing this will allow you to create maps in Dashboards and Visualizations.

Note: Fields that have been given Location categories can be placed within Drill fields to create drillable maps. In order for this to work properly in Dashboards and Visualizations, the Drill Field must be formatted in order of descending geographic size.

For example:
Drill Field
- Country field
- State/Province field
- County field

Map Field
When a Location category is selected, you will also need to specify a Map Field option that matches how the String is formatted.

In order to account for the multiple formatting styles that exist, a number of Map Field options are available.
Map Field Formatting Options

<table>
<thead>
<tr>
<th>Country</th>
<th>Region</th>
<th>State/Province</th>
<th>County Division</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Link to List)</td>
<td>(Link to List)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ISO 3 Letter Country Code</td>
<td>HASC Code*</td>
<td>County/Division Name</td>
<td></td>
</tr>
<tr>
<td>(Link to List)</td>
<td></td>
<td>(Standard names, e.g. ‘Cook’)</td>
<td></td>
</tr>
<tr>
<td>Country Name</td>
<td>Full name</td>
<td>Postal Code*</td>
<td></td>
</tr>
<tr>
<td>(Standard names, e.g. ‘Spain’)</td>
<td>For example: Northwest</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Abbreviated Country Name*</td>
<td>Abbreviated</td>
<td>State/Province Name</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For example: NW</td>
<td>(Standard names, e.g. ‘Illinois’)</td>
<td></td>
</tr>
</tbody>
</table>

*Currently the best way to determine if your data conforms to one of these formats is to browse through the High Charts JavaScript files available here.
Simply find a file for a geographic region that is included in your dataset and then perform a Ctrl-F on the Map Field type of interest to see how values of that format are written.

Data Format for US Regional Map
Full names: Northwest, Southwest, Central, Northeast, Southeast
Abbreviated names: NW, SW, CR, NE, SE

Max Length
Max Length controls the maximum number of characters allowed in values for String fields.
A Data View field’s Max Length must always be greater than or equal to that of the value with the maximum length in a data set. This can be different than the Max Length of a Data Store field.
For example, imagine you have a data set where the longest String in field1 is ‘ReallyBigData.’ This String has a length of 13; so, to properly execute the Data View, field1’s Max Length in the Data View must be 13 or greater.
Like with Precision, you can run into issues if Max Length for field1 has been set to less than 13 in its Data Store - which may lead you to believe that Max Length for the data set is less than 13. Doing so will not cause any issues in the Data Store or an Analyses, but it will cause issues in Data View execution. Namely, a field with too low of a Max Length will throw the following Error in the Executions Log:

Unable to copy data into data store; nested exception is java.sql.SQLException: [Amazon](500310)[Amazon](500310)

If you encounter this error, try raising Max Length in applicable fields and Rebuilding the Data View. Since Data Store field Max Length may actually be lower than true data set Max Length, you may not be able to use this as a reference. If in doubt, there is no harm in setting Max Length to something very high and seeing if this will solve the issue.

Semantic Type
Applying a Semantic Type to any type of field can help to further categorize the field’s content and enable additional functionality throughout Infogix Data3Sixty.

For example, adding the Airport semantic type to a string field containing airport codes could help to categorize a dataset containing unlabeled airport codes.

Displaying URLs as Hyperlinks in Dashboards with Semantic Type
As another specific example, the URL Semantic Type can be applied to String fields that contain String values which should be treated as URLs. Doing so will actually cause the URL field’s Strings to be displayed as clickable hyperlinks in Data Grid dashlets. If the URLs are valid, clicking on these links will open the associated website in another browser tab.

Encrypted Fields
Specifying a Date View field to be encrypted will cause the field’s records to be encrypted in storage.

Permissions Required to Modify Encryption
- Any user with access to a Data View can encrypt a Data View field if it hasn’t yet been encrypted.
- Once a field has been encrypted, only Administrators and Users with Administer or Manage Security permission to the Data View will be able to modify the field’s encryption settings.

Places where you’ll see the effects of Encryption
- An encrypted field that is being visualized may only be used as a Dimension or as a Measure with the None operation.
- Encrypting a Data View field will not cause it to be masked when visualized. For masking, you will need to make the field secure.
Creating Data Views

Data View Properties

Effects on Data when Changing Field Encryption

- If changing field encryption, a migration job will be performed upon save to encrypt/decrypt the Data View’s pre-existing data.

Secure Fields

Specifying a Data View field to be a Secure Field will cause the field to be masked throughout Infogix Data3Sixty. Masking is defined through use of the parameters described below.

Secure Field Parameters

- **Hide this field when user does not have access**
  Users who have access to a Secure Field are those who belong to the Super Groups chosen in the Secure Field’s Super Groups parameter. If this box is checked and a user who is not part of the Secure Field’s Super Groups uses the Data View, they will not see the Secure Field at all.
  If this box is left unchecked, all users will see the Secure Field, but its contents will always be masked.

- **Mask**
  This setting determines how the Secure Field will be masked.

- **Mask Pattern**
  If the User Defined Mask option is chosen, you can use Mask Pattern to define a preset or custom mask for the content of your Secure Field.

- **Display Mask Pattern**
  If the User Defined Mask option is chosen, you can use Display Mask Pattern to define a preset or custom display mask for the content of your Secure Field.

- **Super Groups**
  Users in a Secure Field’s selected Super Groups will be able to see the field with its contents initially masked, as per Mask settings. Unlike other users, however, users within the selected Super Groups have the option to unmask Secure Fields and see their actual contents. Unmasking can be performed throughout the product, at the individual record level or in bulk via Unmask All.
  If a user is not in a Secure Field’s selected Super Groups, they will either see the field as always masked or they won’t see the field at all. This will depend on whether ‘Hide this field when user does not have access’ has been checked.

Permissions Required to Create or Modify Data View Secure Fields

- Any user with access to a Data View can make a Data View field Secure, if it hasn’t yet been marked as secure.

- Once a field has been marked as Secure and the change has been Accepted, only Administrators and Users with Administer or Manage Security permission to the Data View will be able to modify the field’s security settings.
7 Creating Data Views

Data View Properties

Inheriting settings from Data Stores

- If a field is set as Secure, all the security settings will carry over when the field is used in a Data View.
  - When the Secure flag’s setting is inherited, it will not be able to be changed by anyone - unless it is changed in the Data Store.
- Infogix Data3Sixty Administrators and users with Administer or Manage Security permission to the Data View will be able to modify settings besides the Secure flag. Other users will not be able to modify any of these settings, though.

Places where you’ll see the effects of Secure Fields

- Dashboards and the Visualizer

Computed Fields

Computed fields give you the ability to write expressions to manipulate the information held in selected data store fields.

For example, you could create a computed field called Customer Age, which extracts a customer’s birth year from a birth date field and then subtracts that year from the current year.

Infogix Data3Sixty allows for a wide variety of Computed Fields combining fields, operators, functions and/or manually entered values. For additional help on the available functions, see: Additional Function Help.

Syntax when using Fields

Fields used within Computed Fields should be referenced as follows:

{dataStoreName}.fieldName

Where dataStoreName refers to the Data Store the field comes from and fieldName refers to the field itself.

Note that if the field is double clicked within the Expression editor, this syntax will be created automatically.

Drill fields

Drill fields are simply compound fields, built using multiple data store fields, where each field contains more granular information than the field above it.

For example, you might have a Drill field called Customer location. Customer location could contain a Customer Country field, a Customer State/Territory field, and a Customer City field.

When this Customer location Drill field is used in a dashboard or visualization, users will be able to “drill in” and “drill out,” visualizing data at different levels of granularity.
Data View Properties

Group Fields
Group fields are essentially folders, used to group related fields that don’t have a granular relationship. A Group field cannot be used as a field in a dashboard or a visualization; only the fields contained within the group may be used.

Group Fields: Applying Location Category and Map Fields
If you have multiple fields that share the same Location Category and Map Field type, you can use a Group Field to group them together and set these options in bulk.

For example, suppose you had three fields that represented Countries using country names: `vendor_country`, `customer_country`, and `salesrep_country`. Applying a Country Location Category and Country Name Map Field to a Group field would mean that if you placed all 3 of these fields into the Group field, they would all inherit its location settings. Location Category and Map field would not need to be explicitly set for each `country` field.

Geo Fields
Geo fields allow you to group a Latitude and a Longitude field. Latitude and Longitude fields must be of the Decimal type.

When building dashboards or visualizations, Latitude and Longitude Geo Fields can be visualized with the Map charting type as bubbles or as markers.

Date fields
Date fields are simply data view fields that populate with date-formatted data. For example, `Customer Birth Date`.

Load filters
Load filters allow you to filter what is loaded into a data view based on selected criteria. For example, if you had a data store field that contained sales information about all 50 United States but you only wanted to visualize data for Illinois, you could create a Load filter that instructs the data view to only pull sales data for Illinois.

If you don’t create any Load filters, all data view fields will populate with all available data.

When are Load Filters Applied?
Load Filters are applied each time a Data View executes.

Query filters
Query filters allow you to restrict which data from a Data View certain users will be able to access.

For example, if you had a Data View containing a Region field, you could write Query filters that restricted access based on the value of the Region field.
For example, a Northern region Query filter could be assigned to a Northern region Super Group and could have the following Filter Expression:

\[ \text{Region} = \text{`Northern'} \]

Likewise, a Southern region Query filter could be assigned to a Southern region Super Group, with the following Filter Expression:

\[ \text{Region} = \text{`Southern'} \]

With these Query Filters set in place, when users within the Northern Super Group queried the Data View, they would only see records where the value of the Region field was ‘Northern’. Similarly, users within the Southern Super Group would only see records where the value of the Region field was ‘Southern’.

*Allow user to view all data when there are query filters but no matching query filter is found for a user* Checkbox

If this box is *unchecked*, users who were not in either the Northern or Southern Super Group would not see anything when querying the Data View. If this box is *checked*, users who were not in either the Northern or Southern Super Group would see all data.

*When are Query Filters Applied*

Query Filters are applied whenever a Data View is used to build a Dashboard.
Using Profile Fields with Query Filters

If a data set contains a large number of values by which you want to filter, you can avoid writing a separate Query Filter for each one by using Profile Fields.

Profile Fields are values that can be attached to user accounts by your Administrator, and they can be constructed to match the values in a data set.

For example, a set of user accounts could have a State Profile Field attached to them, with a value of 'IL', 'WI', 'FL', or some comma delimited combination of these values depending on the user.
**Creating Data Views**

**Data View Properties**

Within a single Query Filter, you could then use the `PROFILE_FIELD()` function in an expression that checks whether one of the Profile Field values of a user matches the value of a specific field within the Data View.

For example, if the Data View had a `state` field, the following Filter Expression would suffice:

```plaintext
state = PROFILE_FIELD('State')
```

This would ensure that users within the Super Group assigned to this Query Filter would only be able to see records where one of the values of their `State` Profile Field matched the value of a record’s `state` field.

**Using Multiple Query Filters**

When multiple Query Filters are applied, the Query Filters are treated in an OR-like fashion. That is, each Query Filter is considered independently.

For example, if you had the following expressions being used as Query Filters:

1. `field1 > 0`
2. `field2 < 100`

Values such as -1 or 101 would both make it through the Query Filters.

To get an AND effect, you would need to combine these expressions into a single Query Filter.

**Data View System Properties**

If you are an Infogix Data3Sixty Administrator or you have Administer permission to a Data View, you will have access to a System tab when editing Data Views. The System tab provides the ability to create parameters that will override the Execution Sizing settings used when the Data View is executed, as well as the ability to create Execution Properties.

If you need assistance with these settings, please contact Infogix Support.

**Other Data View Properties**

**Data Retention**

This property is used to set a time limit on how long a data view will retain incoming data. It is useful in cases where data becomes irrelevant after a certain amount of time has passed.

**Retention Period** defines the length of time for which a Data View will store data. Data Retention applied at the Data View level will override Retention Period settings at the Environment level made by your Administrator. Time is always calculated from the time data is loaded into a Data Stage.

**Period Precision** defines the precision at which a Retention Period is applied.
For example, if a Data View’s Retention Period was set to 1 year and its Period Precision was set to Month, the Retention Period would span one year back from the current day, plus the remaining days to get to the beginning of the month.

Retention Example 1

For instance, suppose the current day is 6/5/2016.
If Data View Retention Period was set to 1 year and Period Precision was set to Month, the system would first look back one year to 6/5/2015. Then, since Precision is set to Month, the system would go to the beginning of the month, to 6/1/2015. At this point, any data that was loaded into the Data View before 6/1/2015 would be removed.

Retention Example 2

As another example, suppose the current day is 6/5/2016.
If Environment Retention Period was set to 1 year and Period Precision was set to Quarter, the system would first look back one year to 6/5/2015. Then, since Precision is set to Quarter, the system would go to the beginning of the quarter, to 4/1/2015. At this point, any data that was loaded into the Data View before 4/1/2015 would be removed.

Retention Period Over Time
Data Retention is applied once per day, and is calculated from the end of the most recent successful Run or Rebuild. If a Run or Rebuild fails, Data Retention will be calculated from the failed Run/Rebuild's start time.
Retention Period Data Flush

Over time, data will be flushed from Data Views based on the Data Retention settings specified. For instance, in the example above, data from Load 1 on 4/29/2015 is flushed on 5/1/2016. This is because a 1 year retention period with Month precision is used; so, on 5/1/2016, the 4/29/2015 Load falls outside of the calculated Retention period (which spans from 5/1/2015 to 5/1/2016).

Interaction with Data Store Retention
When setting Data View Retention, you also need to consider any Retention settings made to the Data Stores that feed the Data View because these settings may interact.

Data Partition
This property is used to index data into partitions. Partitioning can optimize load time when a data view is queried during data exploration.
Identity Fields
When a Data View contains Secure Fields, Identity Fields are required to ensure the ability to uniquely identify each record in your dataset.

Doing so will protect against cases where the Secure Field was assumed to be the Identity Field, because once a Secure Field is encrypted to a random value, you will no longer be able to use it to uniquely identify records. This is mostly important for auditing purposes, so that the system can accurately track which users have viewed the content of a secure field.

As such, in addition to being required when creating Secure Fields, it is also required that your Identity Field(s) be different fields than your Secure Fields.

Data Load
This setting allows you to specify which data stores to pull from during execution and which parts of those data stores should be loaded.

Default Behavior: *New Data Since Last Load*, for all Data Stores used in the Data View.

Data Load Range Options
All
This setting will pull all data currently residing in the Data View’s Data Stores, whenever the Data View runs. For this reason, *All* works best for static Data Stores.

Avoiding Duplicate Records
Because the *All* setting pulls *all* data currently residing in a Data View’s Data Stores, it can cause unwanted results if you happen to Run the Data View multiple times.

As a simple example, consider a situation where you create a Data Store and load it with 1000 records. If you were to build a Data View using this Data Store, set Data Load to *All*, and then run the Data View once, it would load 1000 records. If you were to then run the Data View a second time, however, it would load the same 1000 records again, on top of the original 1000 from the first run.
When using the All setting, you therefore may need to use the Delete All Data feature to empty your Data View between Runs, to prevent record duplication. Alternatively, you could set Data Load to New Data Since Last Load.

**New Data Since Last Load**

This setting will only pull data from the Data View’s Data Stores that was not present during the Data View’s previous run. For this reason, *New Data Since Last Load* works best for dynamic Data Stores.

As another simple example, consider a situation where you create a Data Store and load it with 1000 records.

If you were to build a Data View using this Data Store, set Data Load to *New Data Since Last Load*, and then run the Data View once, it would load 1000 records. If you were to then run the Data View a second time - without loading any new data into the Data Store - it wouldn’t load anything; however, the original 1000 records would remain.

Alternatively, if you loaded 458 new records into the Data Store and then ran the Data View, it would load those 458 records, on top of the original 1000.
Creating Data Views

Data View Properties

Based on File Path Pattern Parameter

In the diagram above, we have a configuration where a Data Store points to an external file system. When using this Data Store in a Data View, a) you can select a Data Load Range of Based on File Path Pattern Parameter in the Data View, to only pull in files with a file path that matches a specific pattern. After creating the Data View, you will need to create a Process Model. Within the Process Model, you will then need to b) create a variable with the same name as the parameter created in the Data View, within a node that serves as an input to c) an Execute Stage Task that runs the Data View. When the variable is created in part b, you can set the pattern that you want matched as the variable’s value.

After creating such a Process Model, the Data View referenced by the Execute Stage Task will only pull in files from its Data Stores that match the file path pattern set in part b - when the Process Model is used to execute the Data View.
Historical Data Handling

Historical data handling can be used when an identity field is associated to many records that pile up/change day after day - for example, a claim that updates with new data each day.

With such a claim, historical data handling could be used to specify which record to look at, based on date - for example the most recent date available.

**Historical Data Handling Example**
Suppose you have a Data View with the following records:

<table>
<thead>
<tr>
<th>id</th>
<th>date</th>
<th>measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>8/16/2016</td>
<td>1000</td>
</tr>
<tr>
<td>001</td>
<td>8/17/2016</td>
<td>2000</td>
</tr>
</tbody>
</table>

Without Historical Data Handling, this Data View would display 2 records for id 001: one record from 8/16/2016 and one record from 8/17/2016. If however you were to use Historical Data Handling, setting **Identity Field** to id and **Date Time Field** to date, the Data View would only display 1 record, from 8/17/2016. This is because Historical Data Handling displays the most recent record for each Identity Field.

As your Data View executes over time, the most recent record for a chosen Identity Field may change. This setting can allow you to only display that record in Dashboards.

**Use History Uniquing for Queries**
This check box should be checked to enable historical data handling for every query that is performed on the Data View. If the check box is not checked, historical data handling will not be performed.

**Delete Duplicate Data After Load**
This setting allows you to delete any duplicate records in a Data View after the Data View loads. To utilize this feature, you first need to set an Identity Field for the Data View, so that each record within the Data View can be uniquely identified. Within the Delete Duplicate Data After Load field set, you will then need to specify a field to sort by and whether sorting should be performed in ascending or descending order.

When performing duplicate data deletion, the system will group all records by the chosen Identity Field(s), sort by the specified sort field and direction, and then retain the first record in each group.
Executing and Loading a Data View

When a data view is executed, it pulls the most recent data available from the data stores it has been pointed to. Once a data view has been built and loaded, it can then be used to explore data in a dashboard or visualization.

When exploring data it is always important to consider when that data is from. If you aren’t sure, you can always run the data view to populate it with the most recent data available or contact your Administrator for more information.

Running a Data View

First time Run
Running a Data View for the very first time will load its fields with the most recent data available from all of its data stores.
A Data View must be run at least once before it can be used to Explore Data.

Re-Run
Running a data view after it has been run before will update the view’s fields with the most recent data from all of its data stores.
You can use a Data View to explore data during a Re-Run, but visualizations will not display updated data until the Re-Run is complete.

Rebuilding a Data View
Rebuilding a data view will do the same thing as running a data view, however it will also incorporate any changes that have been made to the Data View since the last time it Ran..
Data Views should be Rebuilt after they are modified. More specifically, “modification” may mean:
- The Data Stores the Data View pulls from have changed.
- Data View fields have been added, removed, renamed, or reconfigured.
- Load Filters, Query Filters, or Other Data View Settings have changed.
You cannot use a Data View to explore data while it is being Rebuilt.

Scheduling Data View Executions
Users with Administer permissions to a data view can schedule its executions. If you are familiar with the frequency at which your incoming data will be updating, you can request these permissions from your Infogix Data3Sixty Administrator.
Creating Data Views

Executing and Loading a Data View

Viewing Data View Contents
To view Data View Contents, use the View Stage option available from your Pipeline. Alternatively, you can click the View button present in the toolbar when editing the Data View.

Viewing a Data View allows you to take a look at the amount of records it has loaded during its executions, or its “Contents.” When viewing contents, you can reach each load event’s execution history directly, by selecting the Work ID you are interested in and then using the View Execution button available in the toolbar.

Note that when viewing Data View contents, there will be no files available to download. For this type of content, look to the Data Stores the Data View queries.

Permissions
Permission needed to View: Read
Permission needed to view Executions: Execute

Deleting all Data from a Data View
When editing a Data View, Users with permission to do so can Delete All Data. This functionality should be used when working with a Data View that has previously Run and contains data you are no longer interested in visualizing.

The need to Delete All Data in a Data View may arise when:
- A Data View needs to be Rebuilt.
- A Data View needs to be Re-Run because Data Load settings have changed.

Affect on Dashboards
Before Deleting all Data from a Data View, you should know that doing so will effectively empty out any Dashboards or Visualizations that use that Data View. This means that such Dashboards will be temporarily unavailable and display ‘No Data’ until the Data View is rerun or rebuilt.

To determine if this will be an issue, simply use the Find Usages feature on the Data View while viewing it in a Pipeline.
Executing Data Stages

Once you have built Data Stores, Analyses, Data Views, or Process Models, you will have executable Data Stages.

In Infogix Data3Sixty, an execution occurs any time a Stage acts on data; however, exactly what it means for each Data Stage to “execute” varies by item.

To be able to use what you have built, you should understand what occurs when each Data Stage executes.

Depending on what type of Pipeline you’re working with, deciding when Stages should execute will then be up to your discretion or the discretion of your Administrator.

What it Means for different Data Stages to “Execute”

Run
- Loads the Data Stage with new data based on its Data Load configurations.

Rebuild
- Loads the Data Stage with All data, overriding its Data Load configurations.

Rerun
- Allows you to perform a Run from within the Executions History.

Rollback
- Reverts Data Stage to the state it was in just after its last Run or Rebuild.

Terminate
- Stops a Run or Rebuild mid-process.

More details about specific Data Stage executions are found below.
Data Store
When a Data Store “executes,” it loads new data from its source and places this data into its defined fields.

Knowing when a Data Store loads new data can help you determine: 1) what data you are working with, 2) when to schedule Data View executions, 3) or it can indicate the execution of an Analysis.

4. Rollback
Rolling back a Data Store will restore it to the state it was in prior to its most recent Run. This will delete any data that was added to the Data Store during the most recent Run.

Data View
There are two ways a Data View can execute.

1. Run
When a Data View runs, it grabs data from the Data Stores it has been pointed to and loads the data into its fields, based on the Data View’s Data Load configurations.
2. **Rebuild**

   A Data View Rebuild deletes all data presently residing in a Data View and then loads all data from its Data Stores, overriding the Data View’s Data Load configurations.

   Knowing when a Data View executes can help you determine precisely what data it will allow you to explore.

   Knowing when to perform or schedule Data View executions depends on the frequency at which the Data View’s Data Stores receive new data and also the frequency at which the Data View is modified.

3. **Rollback**

   Rolling back a Data View will restore it to the state it was in prior to its most recent Run. This will delete any data that was added to the Data View during the most recent Run.
Analysis
There are two ways an Analysis can execute.

1. **Run**
   When an Analysis runs, it performs the enhancements and/or analytics defined in its design and then loads its Data Store outputs with new data. The data pulled into the Analysis will be as specified by the Analysis’ inputs’ Data Load configurations.

![Run Diagram]

2. **Rebuild**
   An Analysis Rebuild performs the same activity as a Run - enhancing and/or analyzing data and then loading Data Store outputs; however, all data from all inputs will be pulled into the Analysis regardless of Data Load configurations.

![Rebuild Diagram]

Knowing when an Analysis executes can help you determine exactly what data is contained in a Data Store the Analysis is pushing to - and, subsequently, what data is contained in a Data View that Data Store loads.

Determining when to perform or schedule Analysis execution is primarily dependent upon the frequency at which its Data Store Input receives new data.
3. **Rollback**
Rolling back an Analysis will restore it to the state it was in prior to its most recent Run. This will also delete any data that was added to the Analysis’ Data Store Outputs during the most recent Run.

**Process Model**
Process Models are covered in detail in the next chapter of this guide.
With reference to Executions, a Process Model can be run or rebuilt to carry out the Tasks specified in its design; typically, however, Process Models need only be saved. In most cases, Process Model execution will be triggered by the execution of another Data Stage.

**Process Model**

1. **Run**
   When an Process Model runs, it performs the tasks defined in its design. If Execute Tasks are performed, they will Run the specified Data Stages.
2. **Rebuild**  
A Process Model rebuild performs the same activity as a Run; however, if Execute Tasks are performed, they will Rebuild the specified Data Stages.

3. **Rollback**  
Rolling back a Process Model will restore it to the state it was in prior to its most recent Run. This will also rollback any executables that were triggered by Execute Stages tasks and as a result remove any data that was loaded to Data Stages during the rolled back run.
Rollback Child Processes
When rolling back a Process Model, you have the option to Rollback Child Processes. Filling in this check box will make the rollback additionally rollback any Process Models that were executed by the current Process Model using an Execute Process Task. This rollback of child processes is applied recursively, meaning that if the child processes have child processes, those processes will be rolled back as well - and so on and so forth until all child processes have been rolled back.

Performing or Scheduling Executions
Knowing when to execute a Data Stage is entirely dependent on the role the Stage plays in your data management pipeline.

As an example, consider the following diagram, which illustrates a chain of execution dependencies.
In this diagram, numbered items illustrate points of activity that must occur in sequential order. That is, Data Store A must load incoming data; then, the Analysis can run and output to Data Store B; and, then the Data View can run and load data from Data Store B. In this example - and in any Pipeline - there are numerous ways to implement execution.

**Manual Execution**

For simple Pipelines, with few execution dependencies, manual execution is a viable option. Manual execution is most useful during development, where you can use Run to load the Data Stage based on its Data Load Range settings or Rebuild to override these settings and simply load all data into the Data Stage.

If you have *Execute* permission to an executable Stage, you can run or rebuild it at any time.

**Scheduled Execution**

If you are working with Data Stages that must execute on a continual basis, you can create an execution schedule. This option is useful when working with Data Stores that load routinely and that are connected to Data Views or Analyses.

If you have *Administer* permission to an executable Data Stage, you can schedule its executions at any time.

**Process Models**

When execution dependencies become complex, you can automate executions for multiple Stages and Pipelines using Process Models.

Process Modeling is outlined in the next chapter of this guide.

**Viewing a Data Stage’s Executions History**

Whichever method of execution is chosen, it is important to keep things in sync. To verify that each Stage is executing when it should be, Infogix Data3Sixty provides an Executions History for every executable item.
Depending on which Stage is being viewed, different Executions details will be provided.

**Searching for Executions**

*Filter*

If you are searching for a specific execution, you can use the Filter button that appears within the Executions toolbar, to expand another tool bar containing fields to search by. Fields include: Date Range, Object Type, Status, Process Id, and Work Id.

*Filter By Process Id*

If you are within the Executions screen and you have selected a Process Model, the Filter By Process Id button will become enabled. Clicking this button will filter all Executions in the current Environment by the Process Id of the selected Process Model, showing you all Data Stages that were executed during a specific run of that Process Model.

**Execution Result Codes**

These numeric codes represent the Status of a Data Stage’s execution. They can also be used in Process Model expressions, to create the Model’s logic.

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><strong>Succeeded</strong></td>
</tr>
<tr>
<td></td>
<td>Indicates that the Data Stage has executed without error.</td>
</tr>
</tbody>
</table>
### Executing Data Stages

Infogix Data3Sixty Users can only view the Executions History for one Stage at a time. To do so, select the Stage within its Pipeline and use the Executions drop down in the main toolbar.

Infogix Data3Sixty Administrators can use the Executions button in the main toolbar to view a listing of all Executions that have occurred in a given Environment. Infogix Data3Sixty Administrators may also view Executions Histories individually, in the same way as Users.

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Failed</td>
<td>Something has caused the execution to fail. More details should be available in the Result Detail field.</td>
</tr>
<tr>
<td>9</td>
<td>Lease Expired</td>
<td>Infogix Data3Sixty’s execution engine features an Expired Lease Job, which checks for inactivity due to random error from machine or process crashes. If an execution has crashed, it will be marked with a Status of <em>Lease Expired</em> and a Result Code of 9.</td>
</tr>
<tr>
<td>8</td>
<td>Terminated</td>
<td>Applied to a Process Model that was running when one of its constituents crashed. If the Expired Lease Job finds a crashed Data Stage execution and that Data Stage is part of a Process Model, the Process Model will be Terminated.</td>
</tr>
</tbody>
</table>

**Infogix Data3Sixty Users**

Infogix Data3Sixty Users can only view the Executions History for one Stage at a time. To do so, select the Stage within its Pipeline and use the Executions drop down in the main toolbar.

**Infogix Data3Sixty Administrators**

Infogix Data3Sixty Administrators can use the Executions button in the main toolbar to view a listing of all Executions that have occurred in a given Environment. Infogix Data3Sixty Administrators may also view Executions Histories individually, in the same way as Users.
Creating Process Models

Once you have created Data Stages that need to load or execute on a continuous basis, you can create Process Models to automate these processes.

A Infogix Data3Sixty Process Model is essentially a drawing board, used to orchestrate the flow of data loads and executions between interdependent Data Stages.

The “output” of a Process Model is an automated work flow that can perform a number of tasks and ultimately keep your Data Stages in sync.

Creating a Process Model

As the remainder of this chapter will describe, there are numerous ways to go about building a Process Model.

The general process flow of any Process Model is as follows:

All Process Models begin with a Start Event node, denoted by a thin, circular border. (1) The body of the Process Model will then be comprised of some combination of Intermediate Event nodes (thin, double-circle), Gateway nodes (diamond), and/or Task nodes (square). (2)
Every Process Model then ends with an End Event node, denoted by a thick, circular border. (3)

**Prerequisites**
To create a Process Model, you will need *Create Process Model* and *Write* permissions to a Pipeline. You will also need access to the set of Data Stages for which you’d like to model loading or execution.

In terms of prerequisite knowledge, Process Models require the most comprehensive understanding of an organization’s data and overall Infogix Data3Sixty work flow. Those creating Process Models will need an understanding of all other Data Stages and how they are used to manage and visualize their organization’s data.

**Process Model Variables**
Before delving into the details of each Process Model node, it is important to understand Process Model variables.

Process Model variables are part of every Process Model node. In practice, they can be passed between nodes and used in logical expressions which determine how the Process Model executes.

Exactly how variables are used is dependent on what you want your Process Model to do and which nodes you use to build it; however, there are a few general rules about variables to keep in mind.

**Passing Variables from Node to Node**
When passing a variable from one node to the next, your selection will be limited to “one node prior.” To pass a variable through multiple nodes, you can use Property Reference variables as described below.

**System defined variables**
When a node is used to select an executable Stage, system defined variables come attached to that Stage. These variables are created by Infogix Data3Sixty when the Stage executes or loads.

System defined variables can’t be changed, but they can be passed from node to node and used in logical expressions.

**User defined variables**
User defined variables can be created in-node for custom use. User defined variables have 4 main property types.
Creating Process Models

Creating a Process Model

**Literal type**
The Literal type can be used to create a variable with a fixed value. Literals may then be used by other parts of a Process Model, such as a User Defined Expression variable, or other nodes.

Available data types include String, Datetime, Number, and Boolean.

**Expression type**
Expression variables can be used to apply logic to other variables. Expression variables may use variables from an incoming node, from the current node the expression is being built in, or from the Process Model itself.

In Expressions, variable sources are denoted as follows:
- **input** = Incoming node
- **result** = Current node
- **process** = Entire Process Model

Since multiple variable sources can be used in a single Expression, variables must be prepended by their source using curly braces and a period.

For example:
- `{input}.resultCode` would refer to the system defined result code variable from an incoming node.
- `{result}.literalX` would refer to a user defined literal variable called literalX within the current node.
- `{process}.refEndTime` would refer to the system defined end time variable that exists for the entire Process Model.

**Property Reference type**
Property Reference variables can reference variables from other nodes.

For example, if there were a variable (X) in TaskA that you’d like to use in TaskB, you could create a Property Reference variable (Y). Doing so would effectively pass variable X to TaskB, making it available for use in logical expressions.
Note:
An important restriction to Property References is that they can only be made “one node upstream,” as shown above.

In order to pass a variable through multiple nodes, multiple User defined Property Reference variables must be made.

For example, if there were a variable (X) in TaskA that you’d like to use in TaskC, you would first create a Property Reference variable (Y) in TaskB. Then, you’d need to create another Property Reference variable (Z) in TaskC, referencing Y.

External Type
This type of variable is available in the External Start Event. It is used to reference system level variables that enable specific functionality.

**External Type Variables**

<table>
<thead>
<tr>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>commentLink</td>
<td>Used to access a Dashboard Control’s system generated comment link. Useful when constructing Custom Notification Processes, with a Notify Task.</td>
</tr>
<tr>
<td>thresholdResults</td>
<td>Used to access a JSON object representing a Dashboard Control’s threshold violations. Useful when constructing Custom Notification Processes, with a Notify Task.</td>
</tr>
</tbody>
</table>
Creating variables for use in an Analysis

If a Process Model is to be executing an Analysis, you can access variables that exist in the Process Model within the Analysis through use of the RUNTIME function.

For example, if you wanted access to the Analysis’s workId, you could create a new column within the Analysis using the expression RUNTIME('workId'). Provided the Analysis was being run by a Process Model, the new column would then populate with the Analysis’s workId at runtime.

Execution Result Codes

These numeric codes represent the Status of a Data Stage’s execution. They can be used in Process Model expressions, to create the Model’s logic.

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Succeeded</td>
<td>Indicates that the Data Stage has executed without error.</td>
</tr>
<tr>
<td>1</td>
<td>Failed</td>
<td>Something has caused the execution to fail. More details should be available in the Result Detail field.</td>
</tr>
<tr>
<td>9</td>
<td>Lease Expired</td>
<td>Infogix Data3Sixty’s execution engine features an Expired Lease Job, which checks for inactivity due to random error from machine or process crashes. If an execution has crashed, it will be marked with a Status of Lease Expired and a Result Code of 9.</td>
</tr>
<tr>
<td>8</td>
<td>Terminated</td>
<td>Applied to an execution that either crashed or was manually terminated by a user. If the Expired Lease Job finds a crashed Data Stage execution and that Data Stage is part of a Process Model, the Process Model will be Terminated.</td>
</tr>
</tbody>
</table>
Execution Result Types
Alternatively, Result Type strings can be used in Process Model expressions. Result Types available for use are as follows.

<table>
<thead>
<tr>
<th>Result Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘SUCCESS’</td>
<td>Indicates that the Data Stage has executed without error.</td>
</tr>
<tr>
<td>‘COMPLETED_WITH_ERRORS’</td>
<td>Indicates that the Data Stage has executed successfully but that minor errors occurred. This code can usually be safely ignored.</td>
</tr>
<tr>
<td>‘FAIL’</td>
<td>Something has caused the execution to fail. More details should be available in the Result Detail field.</td>
</tr>
<tr>
<td>‘TERMINATE’</td>
<td>Applied to an execution that either crashed or was manually terminated by a user.</td>
</tr>
</tbody>
</table>

Process Model Nodes
Infogix Data3Sixty’s Process Modeler contains a number of drag and drop nodes, used to model system wide work flows. Below, you can find an overview of each node and its properties.

Event Nodes
Generally speaking, Events are Process Model trigger points. These trigger points will always begin and end a Process Model, and can be placed within a Process Model as well.

Stage Start Event
Stage Start Events are used to initiate Process Models. They reference both a Triggering Stage and an Event Type.

Triggering Stages are Stages that can execute, such as an Analysis or a Data View.

Event Types define whether the Process Model should be triggered when the selected Triggering Stage starts executing or when the execution has ended.

Stage Start Events are the “starting point” in a chain of dependencies. For example, in a situation where you had an Analysis that output newly manipulated data every time it executed and a Data View that used that Data Store Output, your Triggering Stage would be the Analysis and your Event Type would be Execution End.
9 ▪ Creating Process Models

Process Model Nodes

External Start Event
External Start Events are used as starting points in Process Models where process execution is initiated manually, by a User.
This type of Start Event will not initiate a Process Model based on a Stage’s execution; for this functionality, the Stage Start Event is needed.

Stage Intermediate Event
Stage Intermediate Events are just like Stage Start Events, but they are used inside of Process Models - not at the beginning of them. In this way, Stage Intermediate Events act as trigger points mid-process, being routed to by prior nodes and routing to subsequent nodes.

Timer Intermediate Event
A Timer Intermediate Event can be used to trigger Events and Tasks after a specified time delay.
This functionality can be useful if you are working with a Data Stage that has had its executions scheduled. It can also be used to make a Process Model ‘wait’ for Users to perform specified activities or specify how the Process Model should behave if these activities are not performed within a given time frame.

Success Intermediate Event
The Success Intermediate Event can be used to produce output variables to signify success, once a portion of a Process Model has completed.

Error Intermediate Event
The Error Intermediate Event can be used mid-process to throw an error if a specified series of Events/Tasks occur.
Error Intermediate Events have both entry and exit ports, so unlike the Error End Event node, the Error Intermediate Event node will not terminate a Process Model. After the error is thrown, the Process Model will continue on to its next node.

Cancel End Event
The Cancel End Event is used to cancel and rollback all preceding steps in a Process Model.
Cancel End Event nodes can be configured to cancel and rollback preceding steps based on the reception of a variable, such as an Error Code.
**Error End Event**

The Error End Event node will terminate a Process Model’s execution and throw an error code.

Error End Event nodes should be used to terminate an unwanted sequence within a Process Model. For example, if a Gateway expression evaluated to False the Gateway could lead to an Error End Event and terminate the entire Process Model.

**Success End Event**

The Success End Event can be used to end a Process Model that has successfully completed, and also to produce an output variable that signifies Process Model success.

*One Success End Event per Process Model*

When designing your Process Model, you should only include 1 Success End Event. Including multiple Success End Events can cause the Process Model to terminate prematurely, if one Success End Event is reached before other nodes are processed.

**Tasks**

Tasks are units of work to be performed by the Process Model; more informally, they are “the things you want your model to do.” Tasks make up the body of a Process Model and they can be connected by Event nodes or Gateway nodes.

**Execute Stage Task**

The Execute Stage Task node will execute an executable Data Stage. Executable Stages include Analyses and Data Views.

The Execute Stage Task node can be used to automate execution when working with interdependent Stages. For example, if you needed to execute a Data View every time an Analysis executed, you could use the Execute Stage Task node to specify the Data View execution.

*Use Process Reference Time*

Checking this setting will tell the system to use the Ref Time of the entire Process Model when executing the selected Stage, instead of calculating Ref Time for the stage based on its last run.

**Execute Process Task**

The Execute Process Task node can execute another Process Model, from within the Process Model in which it is used.
Execute Asynchronously
When using this node, you can choose to execute the selected Process Model synchronously or asynchronously.

Asynchronous execution will mean that once the Execute Process Task is reached and the external Process Model is triggered, the current Process Model containing the Execute Process Task will immediately move on to its next node - not waiting for the triggered Process Model to finish its execution.

Synchronous execution works the opposite way, i.e. the current Process Model will wait for the external Process Model triggered by the Execute Process Task to finish before it moves onto its next node.

Use Process Reference Time
Checking this setting will tell the system to use the Ref Time of the entire Process Model when executing the selected child Process Model, instead of calculating Ref Time for the child Process Model based on its last run.

Notify from Data Store Task
The Notify from Data Store Task can be used to send a message to a Super Group of users once a data store has finished loading. Users may want to know when a data store load completes, if they intend to use the data store in other Data Stages.

Best Practices for when a Data Store field value might be null
If a Data Store field contains null values for some records and you need to use this field in a Process Model Notify from Data Store task, you can use the COALESCE function in an Analysis first, to find these records in the data store and replace them with a value of your choosing.

You could then output this enhanced data to a new data store and use that Data Store in your Notify from Data Store task.

Notify Task
The Notify Task allows you to create a message to send to a selected Super Group of users, once a series of Events or Tasks have occurred or been performed.

Formatting Messages with HTML
Messages can be formatted through use of common HTML tags.

Syntax for Including variables in Message text
To include a Process Model variable in a Notify task’s message text, use the following syntax, in text: ${variableName}.
For example, a message informing users about a Data Stage’s execution result code could be written as:

*Execution has finished. Result code is: \${resultCode}.*

Depending on the execution’s result, the message sent to users would then either be:

*Execution has finished. Result code is: 1.* or *Execution has finished. Result code is: 0.*

**Process Definition Properties**
- Process Name: \${process.name}
- Process Version: \${process.version}
- Process Display Name: \${process.displayName}
- Process Owning Member Id: \${process.owningMemberId}
- Process Dev Env Id: \${process.devEnvId}
  (Refers to the Environment the Process Model was created in.)

**Runtime Process Properties**
- Process refStartTime: \${process.refStartTime}
- Process refEndTime: \${process.refEndTime}
- Process Id: \${process.id}

**System variables**
Note: These variable names are reserved. If these names are used in user defined variables, the system variable value will be returned.
- User Organization Id: \${orgId}
- User Id: \${userId}
- User: \${user}
- EnvId: \${envId}
  (Refers to the Environment the Process Model executes in.)
- Env: \${env}
  (Refers to the Environment the Process Model executes in.)

**User Defined variables**
To refer to user defined variables, simply use their unique name within the dollar-sign curly brace syntax. For example: \${variableName}.

**Best Practices for when a variable might be null**
If variables included in Notify messages using the above syntax end up null, they will be sent in their raw form to the user, which could cause confusion.

For example, if you wrote the message: *The result code is: \${resultCode}*, this is exactly what would be sent to the user if resultCode was null - simply because the Notify task does not have any other value to use.
To remedy this, create an Expression variable that uses the COALESCE function on resultCode in the Notify task – for example, nullFriendlyResultCode = COALESCE(input.resultCode, 'Execution has not occurred') – and then use that Expression variable in your message instead. For example: The result code is: ${nullFriendlyResultCode}.

With this setup, a null resultCode would create the more user friendly message: The result code is: Execution has not occurred.

Since COALESCE returns its first parameter when that parameter is not null, this setup would still also return 1 or 0 when resultCode held these values.

**Script Task**

The Script Task allows you to create and run a custom script incorporating variables passed from other nodes.

**Using Auto-Complete**

Typing **CTRL+SPACE** in the JavaScript Editor will activate auto-complete. Depending on what else is typed, auto-complete will display Variables and/or Functions with Help Content.

**Input Fields**

Once an input node has been connected to the Script Task’s incoming port, it can be referred to in script using the following syntax:

```
input.fieldName
```

**Output Fields**

To create a new output variable with the Script node, first use the Add button to create a new variable. You will need to assign the variable the applicable data type and initialize it. You can then refer to the variable in script using the following syntax:

```
output.fieldName
```

**Equality type when comparing variables in Scripts (** vs. **)**

When comparing variables in the Script node, it is important to consider which equality type you are using.

**JavaScript Strict Equality (==)**

Values are only considered equal if they are of the same data type and have the same value.

For example, if:

int x = 0, double y = 0.0, and double z = 0.0 then,

x==y is False, whereas y==z is True.

**JavaScript Lenient Equality (==)**
Values of different types can be considered equal.
For example, if:
int x = 0, double y = 0.0, and long z = 0 then,
x== y == z is True.

**Note:** Process Model variables only feature four data types: String, Datetime, Number, and Boolean. This can mean that even when comparing variables of the same data type - for example, Number - you may need to use the Lenient Equality operator (==) to achieve desired return values.

*Javascript’s Numeric Limitations*
When using Javascript it is important to be aware of the language’s limitations, particularly when dealing with numeric fields.

- The maximum numeric integer that can be reliably used in Javascript is 9007199254740991. Any number greater than this may potentially be truncated when using Javascript.
- To avoid truncation of field values greater than 9007199254740991, you will need to tell the Script task to treat a field’s values as Strings.

**Delete From Data Store Task**
The Delete from Data Store task allows you to automate the deletion of content from a Data Store. By default, the node will delete all data from a Data Store; however, other options can be used to retain data that was loaded during specific load events or date ranges.

*Data Store*
This is the Data Store from which to delete data. Only Data Stores of the Internal type should be used.

*Data Deletion*

- **All Data**
  This option will delete all data from the Data Store.

- **All Data Before a Date**
  This option will delete all data from a Data Store that was loaded into the Data Store prior to a specified date. A date can be specified by creating a Datetime variable in the node prior to the Delete From Data Store Task and setting that variable’s value to the desired date. The variable can then be selected as the *Delete Data Before Date Variable*. 
Creating Process Models

Process Model Nodes

- All Data Except for Some Work IDs
  This option will delete all data from a Data Store that was loaded into the Data Store, except for data associated with specific loads, represented by Work IDs. Work IDs can be specified by creating variables in the node prior to the Delete From Data Store Task, or by referencing preexisting variables.

For example:

```
In this example, suppose that the Data Store that the Delete From Data Store Task applies to is the output of the Analysis that the Execute Stage Task applies to. Were this the case, following execution of the Analysis, the Data Store would always only contain content that was loaded during the most recent run.
```

Gateways

Gateways act as routers between Events and Tasks. A Gateway may route to one or multiple Events or Tasks, depending on its type and the Expression specified in its Properties.

Exclusive Gateway

An Exclusive Gateway node has one entry point and two possible exit points. These Gateways lead exclusively to one node, through one of their two exit points.

```
Which exit point the Exclusive gateway chooses is dependent upon the expression written in the node's properties. Expressions apply conditional logic to variables passed into the node from other nodes.
```
**Inclusive Gateway**

An Inclusive Gateway has one entry point and multiple possible exit points. These Gateways can lead to one or multiple nodes, through one or multiple exit points.

With Inclusive Gateways, one exit node is created for each expression that is written. If multiple expressions are evaluated as True, the Inclusive Gateway can route to multiple nodes. If none of the Expressions evaluate to True, the Inclusive Gateway will route to its default node.

*Default Port Behavior*

Inclusive Gateways should always contain enough exit paths to account for all conditions that could occur at that point in the Process Model. This is why they feature a Default port.

If left unconnected, the Default port will terminate the Process Model in Success. The default port can be of more use, however, when it is connected to other nodes and treated like a Fallback Expression.

For example, consider a situation where an Execute Stage Task routes to an Inclusive Gateway, and the Inclusive Gateway has been configured to evaluate an expression. If the expression evaluates to True, the Inclusive Gateway will route to that expression’s exit port. If, however, this expression evaluates to False, the Inclusive Gateway will route to the Default port. Depending on what you would like your Process Model to do, you could leave the default node unconnected as shown above or connect it to another node, as shown below. In this case, a good design would be to route to an Error End Event, to terminate the Process Model with an error.
Parallel Gateway

A Parallel Gateway is used to merge paths from multiple incoming nodes.

If receiving from multiple nodes, the Parallel Gateway will wait for all nodes to finish processing before routing to its outgoing nodes.

Basic Process Modeling Example

We can now illustrate how to use a Process Model by way of a simple example, first introduced in Chapter 5.

In the top window of this diagram, numbered items illustrate points of activity that must occur in sequential order. That is, Data Store A must load incoming data; then, the Analysis can run and output to Data Store B; and, then the Data View can run and load data from Data Store B.
Such a Pipeline Flow can be automated using a Process Model, like the one displayed in the bottom window of this diagram.

- **P.1)** Assuming the Analysis in this example is scheduled to execute on a daily basis, after Data Store A loads, the Process Model begins with a Start Stage Event that specifies the Analysis as its Triggering Stage and Execution End as its Event Type.

- **P.2)** After the Analysis has executed, Data Store B will have loaded. At this point, the Data View can run. This Task is carried out by the Execute Stage Task.

- **P.3** Following a successful Data View run, the Process Model terminates in a Success End Event. The Process Model will be done executing, and won’t do so again until it is triggered by the Analysis’ execution.
Process Model Termination on Failure
An important consideration when creating Process Models is that each Process Model embodies a single process and that this process is designed to terminate if there is a failure at any one node within the Process Model.
Using Multiple Processes within a Process Model

To avoid termination of an entire Process Model due to the failure of a single node when there are other nodes that can continue running in spite of the node’s failure, you can use Execute Process Tasks.

Continuing the example above in the *Process Model Termination Failure* section, consider grouping the Execute Stage Tasks as follows.

These groupings could be refactored into their own Process Models, and then each new Process Model could be added as an additional process to the original Process Model using Execute Process Tasks.
Creating Process Models

Process Model Nodes

Doing so would result in the following Process Model (PM) which references Process Models A and B via Execute Process Tasks.

The advantage of bringing in separate processes via these Execute Process Tasks is that a separate process (here A or B) could fail and it would not result in termination of the entire, original Process Model (here PM).

Such a design, using Execute Process Tasks, is in fact recommended when executing multiple data stages that are not strictly dependent on one another within a single Process Model.
Additional Process Model Settings

*Inherit Reference Time From Start Event*

If a Process Model is triggered by a Start Stage Event, *Inherit Reference Time From Start Event* will set the Process Model’s RefTime to that of the RefTime in the Start Stage Event.

**Failure Notification**

The following Failure Notification parameters can be set to notify users in the event of Process Model failure.

*Groups to Notify on Failure*

This setting allows you to select a Super Group to notify via email if the Process Model fails to execute. Keep in mind that if any executable within the Process Model fails to execute, the entire Process Model will fail to execute. Continuing our example from above, Groups to Notify on Failure would therefore be a useful feature in the case of a failed Data View execution.

**Message**

This parameter can be used to create the subject heading of the failure notification email that will be sent when a Process Model fails.
9 Creating Process Models

Process Model Nodes

Message Details
This parameter can be used to create the message body of the failure notification email that will be sent when a Process Model fails.
Part 2: Data Exploration

Once data has been brought into Infogix Data3Sixty and properly prepared, you can begin to explore and visualize, using Dashboards and the Infogix Data3Sixty Visualizer.

The diagram above depicts how a Data View allows users to create Dashboards in a Pipeline. Data Views can also be explored using the Infogix Data3Sixty Visualizer.

Topics covered in Part Two of this guide include:
- Chapter 10: Building a Dashboard
- Chapter 11: Using the Infogix Data3Sixty Visualizer
Building a Dashboard

Once a data view has been created, you can begin using Infogix Data3Sixty to build dashboards. Dashboards are visual representations of data views, and they can display multiple pieces of information at once in many different ways. Upon completion, dashboards can also be saved and shared for analysis and collaboration.

Where should I build my Dashboard?

A dashboard can be built in a Pipeline or by clicking the Explore Data button. The general process is the same from all starting points, and if you have the proper permissions a dashboard can be saved anywhere regardless of where you begin. For these reasons, where you start building your dashboard is ultimately not as important as where it is saved.

Where should I save my Dashboard?

Where a dashboard is saved affects how others can use it, depending on the permissions assigned to the Pipeline and Path you are working in. Likely, your administrator will have granted you rights to a specific Pipeline and Path where you can build dashboards, but you may also have permission to create Paths.

Pipeline Structure, Data Views, and Permissions

To build a dashboard or use the Visualizer, you will need access to at least one Data View. This Data View can be located in the same Pipeline or Path that you plan to save your dashboards in, or a different location entirely. All that matters is that you have Read and Read Data permission to the Data View, and Create Dashboard and Write to the location in which you intend to save.
A Data View can be used to build Dashboards in the same Pipeline/Path, in a different Path, or in a different Pipeline/Path entirely.

Data View Permissions
To use the Data View, you need Read and Read Data permissions to it.

Dashboard Permissions
To build and save Dashboards, you need Create Dashboard and Write permissions to the location in which you want to Save.

Building a Dashboard for Case Management

If engaged in Case Management, there are other Data Stages that may be used to build a Dashboard; namely, Case Stores and Internal Database Data Stores. Building a Dashboard using a Case Store will allow you to visualize the header records that comprise a case. Building a Dashboard using an Internal Database Data Store, on the other hand, will allow
you to visualize the detail records that are associated to header records in your Case Store. In both scenarios, no Data View is needed; however it should be noted that without the use of a Data View some functionality will be lost.

A Case Store or Internal Database Data Store can be used to build Dashboards in the same Pipeline/Path, in a different Path, or in a different Pipeline/Path entirely.

Dashboard Elements

Infogix Data3Sixty allows you to create a number of Dashboard Elements, otherwise known as dashlets. For a full understanding of each dashlet type, it is recommended that you simply experiment with dashboards. For further instruction, Infogix Data3Sixty’s Guided Tours and Screen Casts.
Charts and Tables

This is Infogix Data3Sixty’s primary dashlet type, which allows you to visualize data in a number of graphical formats. Charts and Tables are built by dragging and dropping fields from the selected data view onto the margins of the plot.

Infogix Data3Sixty dashboards are capable of displaying multiple charts and tables at once, and if they are properly configured charts and tables can interact when the dashboard is used.

Fields

The Fields used in Charts and Tables are configured during the creation of the data view in use. Fields can be dragged onto the x or y-axis, depending on their type.

In general, descriptive fields such as Customer Name or Sales Date can be dragged onto the x-axis.

Numeric fields, such as Total Sales or Order Quantity, are typically dragged onto the y-axis, where they are used to create Computations. Numeric fields can also, however, be dragged onto the x-axis to be used as fields.

Infogix Data3Sixty Charts and Tables are capable of displaying information from multiple fields at once.

Measure Recognition

In Dashboards, numeric fields are recognized as Measures, upon which you can perform computations.

Computations

The Computations used in Charts and Tables are created during the process of building a dashboard. Computations are built using numeric type fields, which Infogix Data3Sixty will automatically identify as Measures within the selected data view.

Computations allow for the manipulation of numeric data using two classes of functions and a Custom Computation option as well:

- **Quick Aggregates**
  These are the aggregation functions that appear when a Measure field is dragged onto the y-axis of a chart. They can be used to obtain a single, aggregate result for every unique record in the selected measure field. For example, Average Total Sales from all Total Sales measures, for each Sales Quarter.

- **None Option**
  The None option listed at the bottom of the Quick Aggregates list allows you to visualize raw data measurements by record without aggregating first. For example, if you were visualizing a measurement by day and used the Average Quick Aggregate, you might get a chart like the following.
If you selected the *None* option, however, your chart would reveal the individual records that were used to calculate each day’s Average during aggregation.

**User Created Computations**

- **Aggregation Functions**
  The Aggregates available via the Computations button are the same Quick Aggregates that appear when a Measure field is dragged onto the left y-axis of a chart. The advantage to creating an aggregation via the Computations button is that it can persist during chart creation.

- **Window Functions**
  Used to obtain a unique result by applying a function to a “window” of rows in the selected measure field.
For example, a *Rolling Sum of Total Sales* for the last four years, where sales values for each month are summed with sales values from the previous three months.

### Custom Computations

Custom Computations allow you to apply Operators and Functions to multiple fields to create customized expressions.

If an aggregation is used in a Custom Computation, you won’t get any Quick Aggregate options when the Custom Computation is dragged onto a chart. Conversely, you can get the Quick Aggregates if you write an expression that doesn’t use aggregation functions.

Additionally, Custom Computations can be used in some Window Functions. If a Window Function allows for this, the Custom Computation will be displayed as a selectable Computation in Field Parameters. As with Fields, Infogix Data3Sixty Charts and Tables are capable of displaying information from multiple Computations at once.

#### Naming convention for User Created Computations

User created computations must use the underscore character _ as the leading character in their Name. For example, _myComputation is a valid Name whereas myComputation is not. If you attempt to name a computation without using this convention, an error will occur.

This restriction does not apply to computation Display Names, however. You can give a user created computation any Display Name you want.

### Card Dashlets

A Card Dashlet allows you to encapsulate multiple Chart/Table Dashlets as cards. Defining activation criteria for the Card Dashlet will then allow the cards to be swapped when filtering is performed on a Dashboard.

#### Adding, Editing, and Deleting Cards

Once you have added a Card Dashlet to your Dashboard, use its top toolbar to add, edit, and delete chart/table dashlet cards within it. Here, adding and editing chart/table dashlet cards works just like adding charts/tables that have been added to your dashboard. The only difference is that the charts/tables will be added to the Card Dashlet and only one will be displayed at a time.
Configuring how Cards are Displayed

Once you have added cards to your Card Dashlet, you can configure how they are displayed by selecting the Card Dashlet itself and clicking the Edit button in the dashboard designer toolbar. Doing so will allow you to define activation fields/values for each card. Whenever a filter is applied to the dashboard that uses the activation field/values specified for a certain card, that card will be displayed in the Card Dashlet.

Activation field/values can be applied to a dashboard via filter controls, brush selection, or drilling.

Default Card

Setting a card to the default card will make it so that it is displayed by default in the card dashlet, without any specific activation field/values applied. Only 1 card may be the Default Card of a Card Dashlet. Additionally, not setting a Default Card will cause the Card Dashlet to display as empty by default.

Activation Field Name and Activation Field Values

As mentioned above, these are the criteria that will cause the Card Dashlet’s display to swap from one card to another.

Activation Field Name is the name of a field from your data source or a statically defined field in a filter control that will cause the card swap to occur. Activation Field Values are the values within the Activation Field that you want to be associated with a particular card and cause it to be displayed.

For example, if you had a card within your Card Dashlet that contained a bar chart displaying measures for a set of customer names, you could choose the CustomerName field as the bar chart’s activation field and use the applicable customer names as the bar chart’s activation values. With such a setup, selecting one of those customer names as a filter for the dashboard would cause the card containing the bar chart to be displayed.

Alternatively, you could use a filter control to define a static field name called ‘Chart Type’ and field values such as ‘US Map’, ‘Sales Bar Chart’, etc. You could then choose ‘Chart Type’ as the bar chart’s Activation Field Name and ‘Sales Bar Chart’ as its Activation Field Value. With such a setup, selecting ‘Sales Bar Chart’ in the dashboard’s filter control would cause the card containing the bar chart to be displayed.

Text Elements

Text elements can be used to add titles or written instructions to aid dashboard users.
Place Holders
You can include a Place Holder in a Text Element, to hold a field from the current data view. The value contained by the place holder will dynamically update based on where you click in your dashboard’s charts.

To include a place holder in a Text Element, reference the placeholder in the text editor using the following notation:

{placeholder number}

The placeholder number will be assigned to the placeholder after it has been built.

Simple Placeholders
Creating a simple placeholder will allow you to apply an aggregation function to a field in your Data View, much like is done when a field is dragged onto the y-axis measure of a chart dashlet.

Computed Placeholders
Creating a computed placeholder will allow you to create a computed field using one or multiple fields, functions, and operators, much like is done when creating a custom field in a chart dashlet.

Filters
Dashboard filters allow dashboard users to selectively filter the data displayed in Charts or Tables, using fields. For example, if you built a chart that displayed Total Sales by Year, you could add a dashboard filter that filtered the data displayed by Country. This Country filter would allow dashboard users to view Total Sales by Year, by Country.

Different field data types will display different filter selection options.

Searchable text fields
String fields used in filters specifically support both literal and wildcard search. Wildcard search can be performed through use of the % character.

Filter Values By Parameter
Filter dashlets can reference a field from your data source or they can be used to create static values.

Filtering by Field Values
This is the default option, which allows you to filter by a field in your data source. Values that populate the filter will be the distinct values found in the selected data source field.
Filtering by Static Values
This option allows you to create a static field with static values. Values that populate the filter will be the values that you add to the Static Values definition.

Static Field Name
This is the name that will be used as the title for the filter dashlet. If you are using this static filter dashlet to activate a card within a card dashlet, this is the name that you should use as the card’s Activation Field Name.

Static Values
This parameter is used to create the static string values that you’d like to use in the filter dashlet.

Filtering by a Dimension List
This option allows you to use actual field names (i.e., dimensions) from the data source you are visualizing when you create a dashboard as the values that populate the filter dashlet.

Dimension List Field Name
This is the name that will be used as the title for the filter dashlet. If you are using this Dimension List filter dashlet to activate a card within a card dashlet, this is the name that you should use as the card’s Activation Field Name.

Dimension List
This parameter is used to select the fields from your data source that you would like to use as filtering values. Note that when filtering, a field’s display name will be displayed; however, the actual value that is used for filtering will be the field’s name. This should be kept in mind particularly when setting up Activation Field Values for a card within a Card Dashlet.

Selection Type
When building a filter dashlet that supports multi-select, you have the option to set the filter dashlet’s Selection Type. By default, this parameter is set to Multiple, which allows users to use the filter dashlet to filter by multiple values at once (using Shift-Click or Ctrl-Click). Alternatively, the Single Selection Type will prevent users from selecting multiple values.

Note that filter negation is not available for Single Selection Type dashlets, as negating a single selection could result in the selection of multiple values.
Cascading Filters

A filter dashlet’s cascading option allows you to enable filtering of the contents of the filter dashlet itself, based on the filter state of your entire dashboard sheet. The filter state of a dashboard sheet is comprised of all filters applied to the dashboard sheet by drilling, brush selection, or filter dashlets.

Off

Filter Dashlets with cascading set to off will always show all of their values as available to select, regardless of other filters applied to the dashboard. This is the default behavior for filter dashlets.

Based on Parent Value in Drill Hierarchy

When a filter dashlet has its cascading option set to Based on Parent Value in Drill Hierarchy, it will consider all filters, besides its own, that have been applied to the dashboard sheet and that come from fields that are its parent in a drill field hierarchy. Cascading will then apply all of these filters from the dashboard sheet to the filter dashlet itself. After cascading, only records that match the dashboard sheet’s filter conditions will remain enabled.

For example, if you had a drill field hierarchy that contained a Grade and a Subgrade field, you could create a filter dashlet with the Subgrade field and set its cascading to Based on Parent Value in Drill Hierarchy. If you then created a chart using the drill field, the Subgrade filter dashlet would cascade every time you drilled past the Grade field in your chart.

For example, if you had Grade values A, B, C, and D and Subgrade values such as A1, A2, A3, etc, drilling in on the A value in your chart would cause the Subgrade filter dashlet to disable all options but those that belonged to Grade A - that is, A1, A2, and A3.

Note that the Subgrade filter dashlet would behave similarly if you created another filter dashlet using the Grade field and made a selection on it.

At the same time, the Subgrade filter would not respond to filtering on fields that were not its parent in a drill hierarchy. For example, if you had another filter dashlet that used an EmployeeId field that was not part of the Grade-Subgrade drill hierarchy, the Subgrade filter dashlet would not cascade at all after making a selection on the EmployeeId filter dashlet.

Global

When a filter dashlet has its cascading option set to Global, it will consider all filters, besides its own, that have been applied to the dashboard sheet. Cascading will then apply all of these filters from the dashboard sheet to the filter dashlet itself. After cascading, only records that match the dashboard sheet’s filter conditions will remain enabled.
For example, suppose you created a filter dashlet using an EmployeeId field, another filter dashlet using a FirstName field, and a bar chart that visualized Salary per EmployeeTitle. If you were to set the EmployeeId filter dashlet to Global cascading, it would cascade whenever you made a selection on the FirstName filter dashlet or performed a brush selection on the chart. For example, if you selected ‘John’ on the FirstName filter dashlet, the EmployeeId filter dashlet would cascade and disable all ids except those that belonged to employees named John. Or, if you performed a brush selection on the chart, and selected ‘Airline Pilot’ as EmployeeTitle, the EmployeeId filter dashlet would cascade and disable all ids except those that belonged to employees with an EmployeeTitle of Airline Pilot.

**Show Relevant Entries Only when Cascading**
This setting will cause only values that are returned by the cascading query to be displayed in the filter control, each time the filter cascades. Turning this setting off will alternatively display all available values within the filter control, however after each cascading query irrelevant values will be sent to the bottom of the filter control’s list and disabled.

**Cumulative Effect of Cascading**
It is important to remember that when a filter dashlet cascades, it will consider all filters that have been applied to the current dashboard sheet, except its own. This means, for example, that if you make multiple selections on multiple filter dashlets, and brush select, and drill, all of these filters may be considered during cascading (depending on whether you’ve chosen Global or Based on Parent Value in Drill Hierarchy).

**Targeting a Specific Dashlet**
Setting this parameter will make it so that selections on the filter control will only be applied to the selected dashlet(s). This type of filter will not be considered part of the dashboard’s global filter collection and it will not show up in the current filter selections list.

**Filter selections**
Filter selections help users navigate and drill through dashboards by listing what fields have been selected and drilled into. They help users identify what information they are looking at and allow for a quick reset, as they can be used to clear selections.

**Resizing Filter Selection Column Widths**
When editing a Dashboard, you may change the widths of a Filter Selection dashlet’s columns. If you Save the Dashboard after doing so, the columns widths will be saved.
Web Content

The web content element allows you to add website access to your dashboard, such as access to a company portal.

Spacers

Spacers add blank space to dashboards. They are helpful for dashboard design and can even be used to leave room for another dashlet if you intend to finish a dashboard later.

Dashboard Sheets

The Add Sheet button allows you to add a new sheet to your dashboard, using the same data view or a new data view.

Advanced Dashboard Features

Once you have a basic grasp of Dashboard Elements, you’ll find that there are a number of advanced features which allow you to further customize the appearance and functionality of your dashboard. Below is a brief introduction to each feature. For a full understanding it is again recommended that you simply experiment with dashboards.

Changing Chart Properties

When building or editing a dashboard, you can alter its colors and fonts by clicking the Properties button. Note that the selected changes will not be displayed until the dashlet is snapped to the dashboard or the chart type is changed.

Changing Chart Display

In addition to the chart options available on the right hand side of the dashboard builder, there is another chart display feature which you can access by clicking on a chart’s Computations tabs. The options available via this feature are dependent on the chart type that is selected.

None Option

When changing chart display, None is an option listed for most chart types. Choosing None will hide the measure from the graph; however, it will still leave the measure available to other features within the dashlet, such as threshold rules and custom sorting.
Drilling to another Data Stage

To drill to another Data Stage from a Dashboard, edit a dashlet within the Dashboard and click on one of its Dimensions. Here you will be given the option to drill to another Dashboard, a Case Store, or an Internal Database Data Store.

Stage
This option allows you to select the specific Data Stage that you want to drill to.

Target in Stage
For Dashboards, this option allows you to select a sheet within the Dashboard to drill to. For Case Stores and Internal Database Data Stores, this option allows you to select a specific search screen to drill to.

Pass Filters
This option allows you to specify whether dimensions from the Dashboard should be passed as filters when drilling to the target. If the target uses the same fields, enabling this option will allow you to filter what is displayed when the drill occurs.

Chart Filters
Chart filters are similar to the Dashboard filters described above in the previous section, however they are not accessible when a completed dashboard is being viewed. Chart filters allow you to filter the information displayed while building a chart, using Fields. This feature would be useful if, for example, a data view contained Sales Records for multiple countries but dashboard users only needed to see records for one.

Importantly, dashboard users will not know a chart filter has been applied unless they are informed by proper labeling.

Multi-Series Charts for Color Coding
If you wish to display multiple descriptive fields on a chart’s x-axis but want the fields to be visually distinguishable by color, you can drag and drop the fields onto the top and bottom x-axis margins of your chart.

When building bar charts, this feature enables more granular color coding and the ability to create stacked and stacked percentage charts.

Applying Thresholds
Thresholds allow you to set a value and value condition which will trigger a change in display for a selected computation and field.
For example, if you are visualizing a data view that loads new data on a daily basis, you could create a threshold. This threshold could be designed to change the bar color of the current Sales Year from blue to red once Total Sales exceed a certain level. If you were working with historical data, you could also use a threshold to highlight areas of a chart that require attention.

Currently, Infogix Data3Sixty features 4 Threshold types: Multipoint, Expression, Western Electric, and Nelson. Each rule is detailed below.

**Features Common to all Thresholds**
When building a Threshold of any type, there are a few features that you will always see.

**Highlight All Points feature**

**What it’s meant for**
Western Electric, Nelson, and Multipoint Threshold Rules are all designed to find sets of consecutive points that exhibit some sort of shared behavior. Point Highlighting allows you to visually identify which points on your graph comprise a set. Depending on the rule that is broken, this set of points may indicate a trend or some problem that is causing your process to go out of control.

**How it works**
When checked, this feature will highlight all points that violate a threshold rule, using the Highlight color - except for the last point, which will always use the Last Highlight Point color. When unchecked, only the Last Highlight Point color will be applied to the last point in a set; however, other points in sets can be highlighted if you hover over the last point in a set with the mouse cursor.

**Process All Rules Feature**

**What it’s meant for**
This feature allows you to apply multiple threshold rules to a single chart.

**How it works**
When checked, this feature will apply all created threshold rules to a chart. When unchecked, only the first threshold rule in your list of rules will be processed. To move a rule to the top of the list, you can select it and then use the Move Up button.
Thresholds that can be applied to most Charting Types

**Single Point Rule**

Single Point Rules are the most basic type of threshold. They allow you to evaluate whether single points within a graph satisfy a custom rule you have created.

For example, say you were visualizing some measure, which ranged in values from 0 to 100, against a time period of 365 days. With a Single Point Rule, you could write a simple rule that found points that were less than 50.

With Single Point Rules, the Last Highlight Point Color is the color that is used to highlight points the threshold rule finds. Additionally, the Highlight Point and Highlight All Points settings are ignored; these settings are however used by other Threshold types.

**Multipoint Rule**

This type of threshold can be applied to any charting type. It allows you to evaluate whether multiple points within a set on a graph satisfy a custom rule you have created.

For example, say you were visualizing some measure, which ranged in values from 0 to 100, against a time period of 365 days. With a Multipoint rule, you could write a rule that, for every 5 days, highlighted series of 3 days where your measure was less than 50.
As pictured above, Multipoint Rules will identify *consecutive* points, but this is not a strict requirement. For instance, with slightly different data, the same Multipoint rule outlined above would still highlight the 3 points out of 5 which satisfied the Point Expression.

**Multipoint Rule parameters**

**Name:** A display name for the Multipoint rule; defaults to “My Multipoint Rule”.

**Drill Fields:** If your chart uses drill fields, you can use this option to specify at which level of the drill view to apply the Multipoint Rule. For example, if your Drill Field contains Year, Quarter, and Month levels, and you specify Month as your threshold rule’s Drill Field, you will only see the application of the rule when you drill into the Month level on your chart.
**Point Expression:** This is the rule that will be evaluated at each point in your graph. You can create it within the Point Expression field, or press the $f(x)$ button to build it in the Expression Editor.

**When __ out of __ points satisfy Point Expression:** This parameter is used to set the number of points that must satisfy the Point Expression in order for points to be highlighted. For example, if you were to specify 3 out of 5 points, highlighting would only occur when, for every 5 points in your chart, 3 points satisfied the Point Expression.

**Aggregate Expression:** This option allows you to create an expression to evaluate in addition to the Point Expression. Aggregate Expressions are applied only to points that first satisfy the Point Expression; this allows you to apply an aggregation function to these points.

As a simple example, we can apply an Aggregate Expression to the initial Multipoint example from above.

Here, the 3 circled points satisfy the Point Expression, but because they don’t satisfy the Aggregate Expression as well they are not highlighted.

**Computation:** This option is used to clarify which computation the Multipoint rule should be applied to. This clarification is necessary because multiple computations can be used in a single chart.

As an example, we can expand our original Multipoint Rule yet again, by adding another Measure to the chart and selecting this measure as the Computation option.
Notice that even though the Point Expression deals with Measure_A, it is Measure_B that gets highlighted - at the points which correspond with the points from Measure_A that satisfy the Point Expression.

**Expression Rule**

This threshold type allows you to apply a single expression to a computation used in a chart. Points that violate the Expression rule will have their color changed to either the Highlight Color or Last Point Highlight Color, regardless of where they fall on the chart.

For example, say you were visualizing some measure, which ranged in values from 0 to 100, against a time period of 365 days. With an Expression rule, you could write a simple rule that highlighted any day where the measure was above 50.
Note: The illustration above assumes that the Highlight All Points option has been checked. If this option is left unchecked, only the final point in the data set that satisfies the Expression rule will be highlighted.

**Expression Rule parameters**

**Name:** A display name for the threshold; defaults to “My Expression rule”.

**Drill Fields:** Functions the same way as in Multipoint rules.

**Expression:** The threshold rule to be applied, built using the expression editor.

**Computation:** This feature functions the same way as described above in Multipoint rules.

### Control Chart Thresholds

The Western Electric Rules and Nelson Rules are process control rule sets invented by statisticians. These rules are only available for selection when a Control Chart is being used, and you are plotting measures against time. Unlike Multipoint and Expression, the Western Electric and Nelson do not require any customization; both are applied through point and click.

#### Western Electric Rules

This rule set was invented by the Western Electric Company and was first published in 1956. It was first applied to achieve statistical control of manufacturing processes. The rule set consists of the four rules described below. Each rule evaluates how points on the control chart fall within three chart zones.
Western Electric Chart Zones

Zone A: Between 2 and 3 standard deviations from the center line mean.
Zone B: Between 1 and 2 standard deviations from the center line mean.
Zone C: Within 1 standard deviation of the center line mean.
UCL: Upper Control Limit
LCL: Lower Control Limit

The 4 Western Electric Rules
Western Electric Rule 1: Any single point falls outside the Upper Control Limit (UCL) or Lower Control Limit (LCL). A point at such a position is over 3 standard deviations away from the center line mean, i.e. beyond Zone A.

Western Electric Rule 2: Two consecutive points out of three, on the same side of the center line, fall beyond Zone B, i.e. beyond two standard deviations from the mean. These points can fall within Zone A or beyond.

*Note: This point also breaks Western Electric Rule 1.
**Western Electric Rule 3:** Four consecutive points out of five, on the same side of the center line, fall beyond Zone C, i.e. beyond one standard deviation from the mean. These points can fall in Zone B, Zone A, or beyond.

**Western Electric Rule 4:** Nine consecutive points fall on the same side of the center line mean, in any Zone.
Nelson Rules

This rule set was first published in the October 1984 issue of the Journal of Quality Technology in an article by Lloyd S. Nelson. Like Western Electric, the Nelson rules are a process control rule set that can be applied to a measure over time. The Nelson rules feature the 8 rules described below, each of which are designed to find trends in a data set. Unlike the Western Electric Rules, Nelson rules do not use the Zone terminology; instead, only standard deviation lines are used.

The 8 Nelson Rules

Nelson Rule 1: Any single point falls outside the Upper Control Limit (UCL) or Lower Control Limit (LCL). A point at such a position is over 3 standard deviations away from the center line mean. This is the same as Western Electric Rule 1.
Nelson Rule 2: Nine or more consecutive points fall on the same side of the center line mean. This is similar to Western Electric Rule 4.

Nelson Rule 3: Six or more consecutive points are continually increasing or continually decreasing.
**Nelson Rule 4:** Fourteen or more consecutive points continuously alternate in direction, i.e. extended oscillation.

**Nelson Rule 5:** Two or three consecutive points out of three, on the same side of the center line, fall beyond two standard deviations from the mean. This is similar to Western Electric Rule 2.
**Nelson Rule 6:** Four or five consecutive points out of five, on the same side of the center line, fall beyond one standard deviation from the mean. This is similar to Western Electric Rule 3.

**Nelson Rule 7:** Fifteen consecutive points are all within one standard deviation from the mean.
**Nelson Rule 8:** Eight consecutive points where no point is within one standard deviation from the mean. These points can be found above the mean and below the mean.

---

**Tool Tips and Overlapping Threshold Rules Violation**

When using either the Western Electric or the Nelson rules sets, it should be noted that multiple rules within these sets can be violated at the same time. For example, see the chart above which depicts Western Electric Rule 2 (but also depicts Rule 1).

This situation can also occur when multiple rule sets are used at the same time. For example, if you were using both Western Electric and Nelson on the same chart, Rule 1 from each rule set would always be applied whenever any point beyond 3 standard deviations from the mean occurred. Depending on the data set, a Multipoint or Expression rule could also simultaneously be broken.

To account for this behavior, Control Charts have a tool tip functionality which will show all rules that are broken at highlighted points. To activate this functionality, make sure the Toggle data tooltips option is on and then simply however over a highlighted point.

**Standard Deviation Date Filter Override for Control Charts**

When using control charts, standard deviation lines are drawn based on the standard deviation of the entire date range present in the visualization. This can be overridden, however, by using the Standard Deviation Date Filter Override option. Like other measure options, this option is available by clicking on the measure tab that is being used in your control chart.
Once configured, the override will redraw standard deviation lines based on the standard deviation of measured values within the specified date range. At the same time, all measured points will still be plotted on your control chart, allowing you to compare points both inside and outside of the override date range to that date range’s standard deviation lines.

**Applying Query Criteria**

Query criteria allow you to customize how a data view is queried and how the results are displayed when a chart or table is built.

Record limits are useful for trimming down or expanding results returned from the data view, and by default they are set to 1000.

**Tip:** Lowering Query Criteria can be particularly helpful when you are working with a large data set, in order to increase chart rendering time.

**Custom Sort**

This parameter allows you to change the order in which data is sorted before it is queried. For example, if you wanted to return the Top 100 SalesAmts, you could use Custom Sort to sort your data set by SalesAmts and then set your record limit to 100.

**Populating a Dashlet Based on Global Filters**

Within Query Criteria, there is a check box labeled: ‘Only populate dashlet when Global Filters are applied, using selected fields’.

When this check box is checked, it will enable a field selector. Moving fields to the Selected column of this field selector will make it so that, when viewing the dashlet to which the Query Criteria apply, the dashlet will only populate with data if all of the Selected fields have been applied as Global Filters to the dashlet’s sheet. Here, Global filter refers to any filter applied to the dashlet’s sheet via filter dashlets, brush selection, or drilling.

If this option is enabled, and not all of the Selected fields are used as Global filters, the dashlet will not populate.

**Setting up a Dashlet’s Targets**

Within a dashlet’s Criteria dialog, there is a Targets tab. This tab allows you to select other Chart/Table dashlets that are present on the same sheet. Designating a dashlet as a “target” will make it so that selections on the current dashlet will only act and be applied as filters on that target dashlet. Multiple dashlets may be chosen as targets. This means that when targets are specified, selections on the current dashlet will not act as “global” filters to every component within the dashboard; these selections will only be applied to the targets.

By default, no targets are set. This will cause selections for the current dashlet to be applied globally.


Include in Filter setting for Dimensions/Series
When clicking on a Dimension or Series button while editing a Chart/Table dashlet, an “Include in Filter” checkbox becomes available. This checkbox controls whether that Dimension/Series is used as a filter when selections are made on the Chart/Table dashlet. If this checkbox is checked, the dimension and its associated value within the selected component will be used as a filter, either globally or in a targeted manner (depending on the dashlet’s Target settings). If the checkbox is not checked, the dimension/series will not be included.

Selection Overlay
When a user clicks on components in a dashboard, filtering will be applied as described above, depending on how the dashlet’s Target settings have been configured. In addition, an overlay will appear on top of the Chart/Table when the selection is made. This overlay will allow the user to:

- “Include” the filter selection, and apply it to the current dashlet.
- “Exclude” the filter selection, and treat it as a negation when applying it to other dashlets in the dashboard.
- “Show Data” with the selection applied as a filter to the current dashlet’s data set.
- “Cancel” to cancel the selection and its filtering.

Note that in cases where a dashlet has been configured to drill to another Data Stage, this overlay will not appear and instead the drill event will just occur.

Tooltip Customization
When building a chart dashlet, you may customize the content that appears in the tooltips that appear when you hover over an element of the chart. Such customization is applied at the measure level. As part of this customization you also have the ability to control whether a dimension, measure, or series is displayed within the axis label of a chart or only within the tooltips of a chart.

Showing Dimensions in Tooltips
You can control whether a dimension is shown in the x-axis of the chart by clicking on the dimension while editing a chart dashlet. Once clicked, a Show only in Tooltip option will be shown. Setting this option to true will remove the dimension from the chart’s x-axis labels and add the dimension’s values to the chart’s tooltips.

Showing Series in Tooltips
You can control whether a series is shown in a chart’s tooltips by clicking on the series while editing a chart dashlet. Once clicked, an Add to Tooltip option will be shown. Setting this option to true will remove the series-based entries from your chart and add the series’ values to the chart’s tooltips.
Showing Measures in Tooltips
You can control whether a measure is shown in a chart or only in the chart’s tooltips by clicking on the measure when editing the chart dashlet. Once clicked, a Tooltip option will be shown. Selecting this option will remove the measure from your chart but display the measure’s values in the tooltips of any remaining chart elements representing other measures.

Customizing Tooltips for Measures
To customize the content that is displayed in a chart’s tooltips, click on a measure while editing the dashlet. Once clicked, a Customize Tooltips options will be displayed. Here you can use a combination of text, placeholders, and HTML to fully control what is displayed in a measure’s tooltip.

Things to know about Tooltip Customization:

- Dimensions, series, and measures are referenced using curly braces. For example:
  {fieldName}

- When a dimension is marked as Show only in Tooltip, it will be automatically added to the chart’s measures’ custom tooltip templates. The default content that will be added for a dimension is of the form: fieldName:{fieldName}

- When a series is marked as Add to Tooltip, it will be automatically added to the chart’s measures’ custom tooltip templates. The default content that will be added is of the form:
  
  {beginRepeat}
  fieldName:{fieldName}
  {endRepeat}

  Here, {beginRepeat} and {endRepeat} create what is referred to as a repeatBlock. Placing a series field name within the repeat block causes every value in the series to be displayed.

- By default, a computationName:{computationName} entry will be present in each measure’s default tooltip content, to refer to the measure itself.

- As mentioned above, clicking on a measure while editing a dashlet and selecting the Tooltip option will hide the measure in the chart. The measure can however be included in the tooltips of other measures by referencing it using its {computationName}.

- In addition to dimensions, series, and measures, different chart types also have different placeholder values that become available when you switch chart types. For example, OHLC charts have: {open}, {high}, {low}, and {close}.

Switching to Source Edit Mode
To customize tooltip content using HTML, switch to Source Edit mode using the Source Edit mode button.
Resetting Custom Tooltip Content
To revert custom tooltip content to the default content for your given dimensions, series, and measures - and their respective Show only in Tooltip, Add to Tooltip, and Tooltip settings - use the Reset button.

Other Dashboard Features

Copying Dashlets
The Copy Dashlet feature allows you to make a copy of any dashlet(s) in your dashboard. To make a copy, select the dashlet(s) and use the Copy dropdown. Once copied, a Clipboard icon will appear in your dashlet palette. You can then drag and drop this icon onto any sheet, to “Paste.”
To select multiple dashlets for copying, use CTRL - CLICK.

Copying Sheets
The Copy dropdown may also be used to copy an entire sheet. Once a sheet is copied, using the Paste Sheet button will create a new sheet that is identical to the copied sheet.

Exporting a Dashboard
The ability to export a dashboard is available throughout Infogix Data3Sixty. Individual dashlets can be exported while building dashboards or while being viewed via the Menu button in each dashlet’s top right corner. Images of full Dashboards can also be exported using the Export button that appears when viewing a Dashboard.

Image export
If you need to share a Dashboard in image format, you can use the Export button to download a .png, a .jpeg, or a .pdf. Keep in mind, however, that these images will only be snapshots of the Dashboard as you are viewing or editing it. Exported Dashboard images do not have interactive elements, like Drill fields and Filters.
Supported formats: PNG, JPEG, PDF

Print View
If you would like to print a snapshot of a Dashboard directly from the browser, use the Print View button, which appears when looking at a Dashboard in View mode.
Delimited File Export
Dashboard data can also be exported, per Dashlet. To do so, use the Menu button in the top right corner of the Dashlet of interest and select Export -> Data or Export -> Data for Excel. After you specify the number of Records to Export from the Dashlet, the download should initiate.

Supported formats: CSV

Undo/Redo when Viewing a Dashboard
When viewing a dashboard outside of Pipelines or Edit mode, an Undo and a Redo button become available in the top toolbar. These buttons will allow you to undo/redo any filters (including those applied by drilling or brush selecting) that have been applied to the dashboard. This functionality can be useful when working with a dashboard that has a large number of filters applied, as it makes it so that you don’t have to remember which filters were applied last as you work with the dashboard.

Refreshing a Dashboard
When viewing a Dashboard in View mode, a Refresh button will become available. This button can be used to refresh all of the Dashboard’s data. Such a refresh may be useful when monitoring a Dashboard that has its data updated on a very frequent basis, since the Dashboard may not be automatically updating its data frequently enough for you.

Unmasking Secure Fields in Dashboards
If you have permission to unmask a secure field, you may unmask it at the record level using the lock icon that appears in records in Data Grids or you may unmask records in bulk using the Unmask All button. Note that every time you perform unmasking, your action will be logged in the product’s Audit Trail, accessible to administrators.

Features Specific to certain Chart Types

Data Grids

Flex Width Columns
When editing a Data Grid dashlet, clicking on a dimension, series, or measure displays a Flex Width option. Setting Flex Width to true for a dimension, series, or measure column will cause the column to expand its width, or flex, to fill any empty space in the Data Grid. When multiple columns are set to Flex Width, they will fill any empty space evenly.
**Lock Columns**

When editing a Data Grid dashlet, clicking on a dimension or series displays a *Lock Column* option. Setting Lock Column to true for a dimension or series column will make the column - and all columns to the left of it - retain position when horizontally scrolling the grid. Using Lock Column can be useful with grids that have many columns that need to be scrolled through but also have a few key fields that always need to be seen in order to identify records.

**Show Tooltip**

When editing a Data Grid dashlet, clicking on a dimension or series displays a *Show Tooltip* option. Setting Show Tooltip to true for a dimension or series will make the column display a tooltip when hovering over its cells. The value displayed within the tooltip will be the same as the value contained in the cell.

This setting can be useful when a column is known to contain extremely long cell values. Setting Show Tooltip to true will make it so that users viewing the Dashboard won’t have to resize a column’s width to read the entire contents of its cells.

**Pivot Tables**

**Custom Measures**

When adding a measure to a Pivot Table, the *Custom* option will allow you to create a custom formula to calculate values for a column. The definition of this formula is performed through use of an Expression Editor - a component used to create expressions throughout the product.

*Custom Measure Syntax*

Custom Measures for Pivot Tables must refer to other aggregate measures already present in the Pivot Table. They must also refer to the aggregate measure using double quotes. For example: `sum("field1AVERAGE")`

*Changing a Measure’s Formula*

Once you have added a measure to a Pivot Table, you can change its Custom Measure formula by clicking on the measure and selecting Pivot Table. Doing so will once again bring up the Expression Editor, where the formula can be defined.

By default, measures that have been added as aggregates will have a blank formula. Adding a formula to an aggregate measure will override the aggregation and cause the formula to be used. Leaving the formula blank on an aggregate measure will cause the aggregation to be used.
Dashboard Settings

Sheet Settings

Show data load timestamp on dashboard
This checkbox adds a latest load time stamp to the top of Dashboards. The time stamp reflects the last time a Dashboard’s default Data View loaded data. This can help Dashboard users verify the last time a Dashboard loaded new data.

Dashboard Dimensions
By default, a Dashboard viewed in View mode will be sized so that every dashlet in the Dashboard fits within a single screen. This can be altered, however, by setting the Min. Width and Min. Height parameters to values larger than those produced when the Use Current Dimensions button is pressed. Doing so will allow for scrollable Dashboards with content that spans more area than a single screen.

Secure Access to Visualizer
This parameter controls whether users can use the Explore a Dashlet in the Visualizer button present on the toolbar of each chart/table dashlet within a dashboard.
When Secure Access to Visualizer is not checked, all users will have access to the Visualizer from all dashlets in all sheets of the dashboard. When Secure Access to Visualizer is checked, only Super Groups specified in the Super Groups parameter will have access to the Visualizer from all dashlets in all sheets of the dashboard. If Secure Access to Visualizer is checked and there are no Super Groups specified, then the Visualizer will not be available to any user.

Styling a Dashboard

Dashboard Fonts

Setting a Global Font Family
Within the Dashboard Settings dialog, you may use the Fonts tab to set a global font family to be applied to all components within your dashboard. A component will use this font family, unless it has had a different font family set at the individual component level.

Styling Font for Filter Dashlets
To style the font for a filter dashlet, edit the filter dashlet and go to the Fonts tab. Here you will find various settings that can be used to control the style of the text elements that comprise a filter dashlet. Note that different options will appear depending on what type
of filter dashlet you are creating. Also note that leaving any font family parameter blank will cause the associated component to inherit the dashboard’s global font family. Setting any font family parameter to a value different than the dashboard’s global font family will act as an override for the associated component.

**Styling Font for Filter Selection Dashlets**
To style the font for a filter selection dashlet, edit the filter selection dashlet and go to the Fonts tab. Here you will find various settings that can be used to control the style of the text elements that comprise a filter selection dashlet. Note that leaving any font family parameter blank will cause the associated component to inherit the dashboard’s global font family. Setting any font family parameter to a value different than the dashboard’s global font family will act as an override for the associated component.

**Styling Font for Chart/Table Dashlets**
Font styling for a Chart/Table dashlet can be customized within the Visualizer Properties dialog, in the Font tab. Here, different options will appear depending on what type of Chart/Table you are creating. Once again, leaving any font family parameter blank will cause the associated component to inherit the dashboard’s global font family; and, setting any font family parameter to a value different than the dashboard’s global font family will act as an override for the associated component.

**Styling Font for Chart/Table Dashlet Titles**
Styling the font for a Chart/Table Dashlet title occurs outside of the Visualizer Properties Font tab. This component can actually be styled using inline HTML.

**Styling Font for Chart/Table Tooltips**
Chart/Table tooltips will inherit the global font family set for the dashboard. For further customization, HTML should be used within the Customize Tooltips dialog for the tooltip’s measure.
Using the Data3Sixty DQ+ Visualizer

You can use Data3Sixty DQ+ to create ad-hoc data visualizations.

Prerequisite: You have created and executed a Data View.

Creating a visualization is very similar to Building a Dashboard. As with Dashboards, the Visualizer allows you to see a visual representation of the data in your data view. If you have already built a dashboard, you will be quite familiar with the features and mechanics of using the Visualizer, as they are essentially the same as dashboard building.

The difference between Dashboards and the Visualizer is in application. Whereas Dashboards require familiarity with a data view, planning, and consideration of design, data exploration with the Visualizer is intended to be much more spontaneous and can be performed by anyone for ad-hoc data exploration. When you are viewing a Dashboard, you can use the Visualizer to investigate a dashlet, as if in edit mode.

To access the Visualizer, click the menu button in the top left corner of the screen and select **Explore Data**.
11 ■ Using the Data3Sixty DQ+ Visualizer

Or, from the Dashboard view, you can open the Visualizer by clicking the Explore a Dashlet in the Visualizer button.

The Explore a Dashlet in the Visualizer button opens the Visualizer in a pop-up window. When you have finished exploring the data, close the Visualizer window to return to the original dashboard.

If you want to save the changes that you have made in the Visualizer, use the Copy To menu to choose whether to create a New Dashboard, or save your changes to an Existing Dashboard.

In general, the Visualizer is a great tool to use if you simply want to experiment with a data view and familiarize yourself with its contents. If you intend to build dashboards and share them with others later on, the Visualizer can be an excellent place to start.

Data3Sixty DQ+ Visualizer Features

Chart filters
Chart filters allow you to filter the information displayed while building a chart, using Fields. This feature would be useful if, for example, a data view contained Sales Records for multiple countries but you only wanted to see records for one.

Thresholds
Thresholds allow you to set a value and value condition which will trigger a change in display for a selected computation in a selected field.

When using the Visualizer, thresholds can help you highlight areas of a chart that fall below or exceed certain criteria. This can be particularly helpful if you are visualizing numerous records at once.

The details of creating Thresholds are covered in the Dashboards section of this guide.
Query criteria
Query criteria allow you to customize how a data view is queried and how the results are displayed when a chart or table is built.

Record limits are useful for trimming down or expanding results returned from the data view, and by default they are set to 1000.

Custom sort fields allow you to change the order in which information is displayed.

Show Data Details
The Visualizer’s Show Details feature allows you to select fields to display in a table when a chart is clicked. Details can include information that is not readily apparent in the visualization alone.

For example, if you were visualizing Sales Dates vs. Total Sales Amounts, you could set Show Details to display a table that included a breakdown of sales amounts per date. If you wanted to view more information without making your chart any more complex, you could also add Ship To Country and Customer Name fields to the table’s details. Upon clicking the chart, you’d then be presented with a table displaying records with information from all chosen fields.

If you are using non-drillable fields, Details will display upon clicking any section of your chart. If you are using drillable fields, Details will display when you reach the bottom level.

Customizing Visualizer Properties
This section covers the modifiable Properties available when building charts, which include things like colors and formatting for text values. Note that the content covered here also applies to any chart created in a Dashboard.

Also note that Properties tabs are context sensitive per chart, meaning that different options are available for different chart types.

Color Tab

Color Palette
This list of colors allows you to define chart colors by the order in which elements appear on the chart. Application can vary per charting type. For example, with Bar charts, each color selection is applied to each new series in your bar chart; whereas with a Pie charts, each color selection is applied to each slice of your pie.

Threshold Color Range
Not to be confused with the Thresholds feature, this setting defines the color gradient used for charts that implement gradient shading, such as Geographical Maps, Heat Maps, and Tree Maps.
Chart Tab

Show legends
This option allows you to toggle the color coded legends that appear at the bottom of your chart.

Align Ticks
This setting is relevant when multiple measures are visualized in a chart that uses an x and y axis.

Unaligned Ticks
If left unchecked, each measure on the y-axis will display its own tick marks. This may be useful in cases where it is important to emphasize that the measures are using different units of measure or scale.

Aligned Ticks
On the other hand, when Align Ticks is checked, measures on the y-axis will share tick marks. This setting will lead to charts that are less cluttered, and it may be useful when multiple measures are being displayed.
Numeric symbols
These symbols allow you to configure how measurements are labeled in your charts. The default values are commonly used metric prefixes.

<table>
<thead>
<tr>
<th>Metric Prefix</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>k</td>
<td>Represents thousands; so, for example, a value of 10,000 will be labeled as 10k on the axis of a chart.</td>
</tr>
<tr>
<td>M</td>
<td>Represents millions; so, for example, a value of 1,000,000,000 (one billion) will be labeled as 1,000M on the axis of a chart.</td>
</tr>
<tr>
<td>B</td>
<td>Represents billions, so, for example, a value of 1,000,000,000,000 (one trillion) will be labeled as 1,000B on the axis of a chart.</td>
</tr>
<tr>
<td>T</td>
<td>Represents trillions, so, for example, a value of 1,000,000,000,000,000,000 (one quadrillion) will be labeled as 1,000T on the axis of a chart.</td>
</tr>
<tr>
<td>Q</td>
<td>Represents quadrillions so, for example, a value of 1,000,000,000,000,000,000,000,000 (one quintillion) will be labeled as 1,000Q on the axis of a chart.</td>
</tr>
</tbody>
</table>
While generally suitable for most applications, these labels can be changed. To do so, you can enter replacement values, while making sure that all potential values still have a label. For example, if you wanted to change \( k \) to \textit{thousand}, you could enter: \textit{thousand, M, B, T, Q} into the Numeric Symbols field.

**Axes Tab**

This tab contains a number of settings to allow for granular control of how Axes are displayed in your chart. For a complete understanding of these features, it is recommended that you experiment with them while building your charts, to see what effects they will have.

*Commonly used Axes tab features*

- **Making 2 or more Measures Share an Axis**
  Select each measure and check the \textit{Use Shared Axis} box.

- **Changing Dimension display**
  To change the formatting and content of all Dimensions in your chart, select the Dimension option in the left hand panel.
  - Changes within the Axis properties will alter the way values are organized and the content/display of the Title for your x-axis.
  - Changes within the Labels properties will alter the way discrete categories or values within your Dimension(s) are organized and displayed.

- **Changing Measures display**
  To change the formatting and content of individual Measures in your chart, select a Measure in the left hand panel.
  - Changes within the Axis properties will alter the way values are organized and the content/display of the Title for your Measure on the y-axis.
  - Changes within the Labels properties will alter the way the values for your measure on the y-axis are organized and displayed.
Using the Data3Sixty DQ+ Visualizer

Measure Labels

Dimension Labels

myDimension

Measure Title

Dimension Title

Measure

Labels

Dimension

Title

 categoryA

 categoryB

 categoryC

 Dimension Labels

Dimension Title
- **Tick Interval**
  On chart types that support a y-axis or linear x-axis, the numeric increment between tick marks may be controlled via the ‘Tick Interval’ setting. By default, these tick mark intervals are automatically decided by the system.
  For example, consider the following default rendering of a bar chart:

  ![Default Bar Chart](chart1.png)

  Setting the Tick Interval for `measure1` to 50 would have the following effect:

  ![Tick Interval Example](chart2.png)

  **Note:** When modifying Tick Interval, it is important to note that your value will only be used if there is enough room on the chart to render the numbers and lines. If there is not enough room, the system will automatically assign the smallest possible Tick Interval. This will often occur when a very small Tick Interval is specified on a chart with a large range of values.
Using the Data3Sixty DQ+ Visualizer

- **Using Formatters**
  The default value for the Formatter field for charts that use x and y axes is `{value}`. To add additional formatting to your display, you can add different symbols to `{value}`. For example `${value}` would display $ dollar signs in front of each value displayed.

**Grid Style Tab**
This Visualizer Properties tab becomes available when Pivot Grid and Data Grid charts are in use. It will allow you to customize a number of settings that are unique to grids.

- **Drill Type**
  When drilling from a Data Grid, the default behavior is for values from every cell in the row to be passed as filters. Drill Type gives you the option to maintain this default behavior and ‘Drill By Row’, or only pass the specific value of the cell that was clicked, to ‘Drill By Cell’.

- **Stripe Rows**
  This checkbox controls whether alternating grid rows are striped.

- **Autonumber Column**
  This checkbox controls whether an autonumber column is displayed as the first column in the grid.

**Style Tab**
This Visualizer Properties tab becomes available when Line and Spline type charts are in use. It will allow you to change the line style and fill of such charts.

- **Plotting Nulls as Zeros**
  When building Line and Spline charts, you can plot measures where the value is null as zero by finding the measure within the Style Tab and then checking the Plot Nulls as Zeros check box. Doing so will allow you to plot continuous Line/Spline charts despite there being missing values in your data.

**Data Label Tab**
This Visualizer Properties tab becomes available when Pie or Funnel type charts are in use. It will allow you to alter the content and display of the text that label sections of such charts.

- **Using Formatters**
  The default value for the Formatter field for Pie and Funnel type charts is `{point.name}`. To add additional formatting to your display, you can add different symbols to `{point.name}`. For example `${point.name}` would display $ dollar signs in front of each value displayed.
Header Tab
This Visualizer Properties tab becomes available when Data Grid type charts are in use. It allows you to change the names of column headers in such charts.

Gauge Tab
This Visualizer Properties tab becomes available when Gauge type charts are in use. It allows you to change the Title of such charts. Additional Gauge customization is available by clicking on the Gauge’s measure on the y-axis.

Important Notes about the Infogix Data3Sixty Visualizer

Visualizations are not saved
The Visualizer is meant to be used as a quick, what-if tool, and visualizations are meant to be much simpler than dashboards. Visualizations will temporarily save per Infogix Data3Sixty session but when you log off they will disappear.

Saving a Visualizer State to a Dashboard
Once you have constructed a valid chart, a Copy To button will become available for use in the Visualizer’s toolbar. This button can be used to copy your chart to a new Dashboard or an existing Dashboard, to be saved.

Note that when copying to an existing Dashboard, the copied chart will appear as a copied dashlet palette item in the Dashboard designer palette. You will need to drag and drop this item onto one of your Dashboard’s sheets in order to copy over your chart.

Charts and Tables Only
Visualizations can only include charts or tables. To include web content, text elements, filters, or filter selections, you’ll need to create a dashboard.

1 Sheet, 2 Chart Limit
Visualizations are limited to 1 sheet and 2 charts. The 2 chart capability is enabled by clicking the Split View button in the Visualizer toolbar.
Part 3: Sharing, Collaboration, and Controls

Once Data Stages have been created, they can be shared with other Infogix Data3Sixty users to enable collaboration. This section of the User Guide covers both how to share your Data Stages and how to make use of Infogix Data3Sixty’s collaboration features.

The general steps needed to share and collaborate in Infogix Data3Sixty are depicted above; however, details will vary depending on where the Data Stage is saved.

This section of the guide covers:
- Granting permissions
- Assigning Watchers
- Annotations, Comments, Controls, and Attachments
- Favoriting a Dashboard
Permissions and Collaboration Features

In Infogix Data3Sixty, Data Exploration includes the ability to share Data Stages to collaborate with other users. To share a Data Stage, you will need to understand the basics of permissions and the concept of Adding a Watcher. To collaborate, you can use: Annotations, Controls, Attachments, and Comments.

Sharing Data Stages

After creating and saving a Data Stage, you can share it with other users. Depending upon where the Data Stage is saved and your User permissions, the process of sharing will vary.

Sharing in Pipelines

In most cases, the delegation of rights to Data Stages will be managed solely by your administrator - meaning that once something has been created, the proper group of users will have access to it.

In some cases, however, you the user may be given Administer permissions to certain Data Stages. In these cases, you will need a basic understanding of permissions, which are covered below. For a more detailed treatment of permissions, you can also reference the Understanding Pipelines section of the Infogix Data3Sixty Administrator Guide.
Who to Share With: Super Groups
If you’ve been given Administer rights to a Data Stage, you’ll have the ability to grant permissions to Super Groups. Super Groups are simply groups of users to which you can collectively delegate access rights.

Likely, your administrator will have given your Super Groups meaningful names - so you should be able to figure out which users belong to which Super Groups. If this is not the case, you will need to contact your Administrator to find out which users are in which Super Groups.

How to Grant Permissions
When sharing a Data Stage, the following permissions can be applied, by right-clicking the Data Stage, selecting Edit Settings, and then entering the Security tab. The listed permissions can then be granted to Super Groups.
### Pipeline Permission/Action Matrix

<table>
<thead>
<tr>
<th>Permission</th>
<th>Can be applied to:</th>
<th>Type</th>
<th>Allows:</th>
<th>Implies:</th>
</tr>
</thead>
</table>
| Administer | Any Stage          | Data permission | - Access to the Stage  
- The right to grant Stage access to other Super Groups.  
- The right schedule Data Stage execution.  
- The right to view Data Store Load Events in the Executions History. |
|            |                    |      | Note: A User with Administer permissions to a Data Stage is different than a Infogix Data3Sixty Administrator. | Read |
| Create [Stage] | Pipeline, Path | Functional (UI) permission | The right to create the selected Stage.  
Must be combined with Write to Save the Data Stage.  
Users will need Read Data to a Data Store to Create Analysis or Create Data View.  
Users will need Read Data to a Data View to Create Dashboard. |
| Read       | Any Stage          | Data permission | The right to view the selected Stage. |
|            |                    |      | N/A     |
### Permissions and Collaboration Features

**Sharing Data Stages**

<table>
<thead>
<tr>
<th>Permission</th>
<th>Can be applied to:</th>
<th>Type</th>
<th>Allows:</th>
<th>Implies:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>Any Stage</td>
<td>Data permission</td>
<td>■ The right to make and save changes to or within the selected Stage.</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>■ The right to create Paths</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><em>If applied w/out Create, will grant the right to modify existing Stages.</em></td>
<td></td>
</tr>
<tr>
<td>Change Data</td>
<td>Any Stage</td>
<td>Data Permission</td>
<td>■ The right to Upload Files to Data Stores.</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>■ The right to Delete all Data from Data Stores and Data Views</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>■ Required for Case Worker to work Case Store.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>■ Required to modify records in Data Entry.</td>
<td></td>
</tr>
</tbody>
</table>
Permissions and Collaboration Features

Sharing Data Stages

<table>
<thead>
<tr>
<th>Permission</th>
<th>Can be applied to:</th>
<th>Type</th>
<th>Allows:</th>
<th>Implies:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read Data</td>
<td>Any Stage; most applicable to Data Stores and Data Views.</td>
<td>Data Permission</td>
<td>- The right to download Data Store contents.</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- The right to use Data Store contents in an Analysis or Data View, when applied to a Data Store.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- The right to see Data View contents in a Dashboard, when applied to a Data View.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Required for Case Worker to work Case Store.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Required to view records in Data Entry.</td>
<td></td>
</tr>
<tr>
<td>Manage Security</td>
<td>Any Stage</td>
<td>Functional (UI)</td>
<td>- The right to delegate permissions to a Data Stage to other Users.</td>
<td>Read</td>
</tr>
<tr>
<td>Execute</td>
<td>Any Stage; most applicable to Analyses, Data Views, and Process Models.</td>
<td>Functional (UI)</td>
<td>- The right to Run or Rebuild executable Data Stages.</td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td></td>
<td>permission</td>
<td>- The right to view an Executable Data Stage’s executions history.</td>
<td></td>
</tr>
</tbody>
</table>

Permissions Applied at Multiple Levels
As you share and as you work with Data Stages that have been shared with you, permissions may get complicated. Fortunately, they will usually be managed by your administrator.
Some important Things to Remember

- Within an Environment, permissions can be assigned at the Pipeline, Path, or individual Data Stage level.
Permissions and Collaboration Features

Sharing Data Stages

- Your organization is likely operating within multiple Environments, Pipelines, and Paths. You may or may not have access to all of them - so there are likely things operating behind the scenes that you don’t know about.

For more on Permissions, see the Understanding Pipelines section of the Infogix Data3Sixty Administrator Guide.

Moving, Copying, or Duplicating Data Stages

Once Data Stages are created they can be Moved, Copied, or Duplicated to destinations throughout an Environment.

Permissions required to Move
Administrator user or regular User with Write permission to source location and Write + Create [Stage] permissions to the destination.

Permissions required to Copy or Duplicate
Administrator user or regular User with Read permission to source location and Write + Create [Stage] permissions to the destination.

Moving a Data Stage
Moving a Data Stage is the simplest of the three options mentioned above, as it simply entails relocating a Data Stage from one place to another. Moving can be performed by dragging and dropping, or right-clicking.

Copying a Data Stage
Copying a Data Stage creates an independent copy of the selected item.

Things to know:

1. Copying an executable Data Stage (Analysis, Data View, Process Model) does not copy the original stage’s Executions History. Newly copied executables will have Ran zero times.
2. Copying an external Data Store created from file will only copy the Data Store’s definitions - not its data. A newly copied external Data Store created from file will be empty.
4. If Copying to the same Path or Folder, _Copy# will be automatically appended to a Data Stage’s name. If Copying to a new Path or Folder, _Copy# will not be appended. To avoid confusion, it is recommended that you give copies unique names.
5. **References and Usages:** Newly copied Data Stages will reference and use the same Data Stages as originals. This means that caution is required when copying executable Data Stages. For example, if you copied an Analysis and modified the copy’s design but didn’t change the Data Store Output, you would end up with two Analyses pushing to the same Data Store Output. To avoid this situation, use Duplicate.

### Duplicating a Data Stage

Duplicating a Data Stage creates an independent copy of the selected item, and also gives you the option to copy Data Stages the selected item references or uses. This can be particularly helpful when you want to make a copy of a Data Stage and modify it without affecting the Data Stages it is connected to.

For example, you could Duplicate an Analysis and its Data Store Output so that you could modify your duplicate Analysis and run it without affecting the original Data Store Output.

**Things to know:**
- Bullet points 1-3 from the Copying a Data Stage section above also apply to Duplication.
- Duplication requires you to give the duplicate a unique name.

### Assigning Watchers

Once other users have permissions to your Data Stages, you can facilitate collaboration by making them those users Watchers.

The Watcher of a Data Stage is automatically notified via email whenever an annotation, comment, or attachment is made. To add a Watcher, select any Data Stage and then use the Watch drop down available in the toolbar.

**Who Can Watch: Users and Groups**

Either individual users can be made Watchers or Groups of users can be made Watchers. Note however that Groups are not the same thing as Super Groups.

Groups are primarily a tool used by Administrators to ease the creation of Super Groups; however, they can also be used by regular users when using the Watcher feature. Here, Groups are used instead of Super Groups because there may be users who are part of a Super Group that may not necessarily need to Watch the given Data Stage.

If this Super Group is composed of Groups you can then just select a section of it for Watching.
Again, like Super Groups, your organization’s Groups may have been given meaningful names by your Administrator to clue you in as to which users belong to them, but if not you may need to contact your administrator to find out which users are part of which Groups.

**Collaboration Features**

Infogix Data3Sixty’s Collaboration features allow users to communicate and share external files related to Data Stages. As an added benefit, all collaboration is saved for future reference.
Note:
If a user has been granted at least Read rights to a Data Stage, they will be able to use Infogix Data3Sixty’s collaboration features.

Dashboard Annotations
The annotations feature allows you to highlight sections of dashboards and add written notes.

For example, if you were viewing a dashboard and you noticed that Total Sales were much lower than average in a certain month, you could use the annotations feature to highlight that month and add a note, such as: “Why were sales so much lower here?”

This annotation would act as both a personal reminder to investigate the situation later, and an inquiry to other Infogix Data3Sixty users who were Watching the dashboard.

Attaching Documents
The documents feature allows you to attach external files to any Data Stage. This allows users to perform file sharing and supplement their Data Stages with more information.

For example, if you were investigating a period of time on a dashboard during which Total Sales were low, you could make an annotation highlighting this time period and making a request for more information. In response, other dashboard watchers could attach sales strategy documents from that time period.

As another example, users unfamiliar with creating Data Stores in Infogix Data3Sixty could take advantage of the attachment feature to share a file containing data. Another user more familiar with Data Store creation could then download that file to their local computer and use it to create an External Data Store.

Making Comments
The comments feature allows you to add written notes to Data Stages.

A comment might be made in response to an annotation, another comment, or an attachment; or, it could simply act as a less specific form of a dashboard annotation.

Comment Threads
Infogix Data3Sixty comments also include a Thread feature, which allows users to view conversation threads in isolation.

To use the Thread feature, toggle on the Threads button, select a thread, and then click the Details button. Doing so will bring up a new window listing all comments within the selected thread.
Pinning Favorite Dashboards to your Home Screen

If you plan on working with a dashboard repeatedly, it can be convenient to Favorite the dashboard and pin it to your Home screen.

To Favorite a dashboard, simply open it and click the Favorite button in the toolbar. Doing so will pin the dashboard to your Home screen. If you pin multiple dashboards, they will appear underneath the Home screen’s main toolbar as tabs.

Customizing the Explore Data Menu

When clicking the Explore Data Menu, a More... option is displayed. When More... is selected, a listing of all Dashboards you have access to will appear. Selecting a Dashboard will open it for viewing and also append it to the Explore Data Menu for quick access. As you view more Dashboards, More... will display up to the 5 most recent ones that you have viewed.
Dashboard Controls

Controls are Thresholds that are built in the Collaboration panel, while viewing a Dashboard in a Pipeline. Thresholds applied at this level are called Controls because they have additional functionality that can help you operationalize your data processing Pipelines.

If you have already read the Applying Thresholds documentation or built a Threshold, you already have a basic understanding of how Controls work.

The content of this section therefore focuses on what Controls have to offer that ordinary Thresholds do not; and, how to determine when you should use a Control instead of simply building a Threshold into your Dashboard during initial creation.

Notification Functionality

Controls allow you to notify specific groups of users when a Threshold Rule is broken. Like Dashboard Annotations, this notification is sent via email and it will contain a link to the Dashboard containing the broken Control.

The links sent in Control notifications are designed to take the recipient directly to the “point” on the Dashboard where the Control’s Threshold rule was broken. With dynamic data, this “point” will represent the state of data at some time in the past. Also like Annotations, these points are recorded as Comments; so, every time a Control’s Threshold rule is broken, a Comment highlighting the violation is made. Such comments allow users to return to points in time where Controlled Threshold rules were broken.

A comment created by the failure of a Control will be made by: Infogix Data3Sixty System.

Control Parameters

Display Name: The name that will be used to identify the Control in the Collaboration panel.

Comment Text: This is the text that will be used when the Control is triggered and a Comment is created in the Collaboration panel, to show where the Control’s Threshold rule was broken. Comment Text should therefore include information that will help the person who has been notified. For example, notes about the Threshold rule, why it is an issue that it has been broken, and what steps to take next.
Visible To Group: This parameter defines which Super Groups can see this Control listed in the Collaboration panel. It also defines who is emailed when the Control’s Threshold rule is broken and who will be able to see the comments made by the Control.

Threshold Rules: When building Controls, you have the ability to build a Threshold for any chart dashlet in a Dashboard. Once a dashlet is selected, Thresholds are built the same way they are built within Dashboards. Note: If your Dashboard has multiple sheets, you must add the Control while viewing the sheet of interest to have that sheet’s dashlets listed in the Threshold Rules parameter.

When are Controls applied?
If you’ve ever built a Threshold inside of a Dashboard, you probably noticed that it was immediately applied - and that any points that broke the Threshold rule were immediately highlighted. This is not the case with Thresholds built inside of Controls.

Control Thresholds are evaluated each time a Dashboard’s Data View is Run*, i.e. each time new data is brought to the Dashboard. Depending on how you have scheduled your Data View, this functionality can allow for real time monitoring of the Dashboard’s Controls.

To see this in action after building your Control, you will need to Rebuild the Data View used by the Dashboard. If any points that break the Control are found during the Rebuild, users will be notified. Additionally, after the Rebuild, any time the Data View loads new data, the Control will be evaluated.

*Note: A normal Threshold will also be applied to new data after a Data View Run, but the Threshold won’t create comments and notify users.

Viewing Control Comments
Whenever a Control rule is broken, a new Comment will appear in the Comments section of the Dashboard’s Collaboration panel. Like a Dashboard Annotation, this comment represents a snapshot of the point in time when a Control rule was broken. Clicking on this comment will highlight the relevant points that broke the Control.

Additionally, a broken Control will trigger an email notification containing a direct link to the newly made Comment. This link will bring the recipient of the email directly to the points on a chart where the Control threshold rule was broken.

How Many Comments Are Made?
Each time a Data View loads new data, one comment is made for all points that broke the Control at the time of the load. Accordingly, just one email per user is sent for that Comment.
**Basic Control Flow**

Once a Data Store to Data View to Dashboard relationship has been established, it is likely that the Data View will be loading whenever the Data Store receives new data. In the simplest of scenarios, this would mean that whenever you loaded new data into the Data Store and then ran the Data View, points that broke a Control would be highlighted on the Dashboard, accompanied by a Comment and user notification.
Controls’ Interaction with Other Data Stage Configurations

There are, however, a number of configurations at other Data Stage levels that can prevent newly loaded data from making it to a Dashboard and from being considered by a Control. One simple example is Dashboard Query Criteria. If the Dashboard is only configured to display 1000 records, and your newly loaded data is the 1001th record in its Data Store, this record will not trigger the Control - even if the record breaks the Control’s Threshold rule.

There are a number of Data Stage configurations that can prevent data from being considered by a Control, including:

- Data View Load Filters, Query Filters, and other settings made in the Other tab
- Dashboard Query Criteria and Filters
- An Analysis that alters a Data Store that is used by a Data View

It is therefore important to consider all of these possibilities when writing a Control.
Control Permissions
A user must have *Write* permission to a Dashboard in order to Add or Edit a Control. Only users who are within the Control’s *Visible to Group* will be able to see the Comments the system has made when the Control has failed.

When to use a Control instead of just a Threshold
In sum, Controls extend the functionality of Thresholds by making Thresholds more collaborative and automated.

A use case for a Threshold instead of a Control could be if you are looking at a large set of data just once - and you want to be able to quickly identify points that break some rule.* Conversely, a use case for a Control could be any type of Threshold use case where it might also help to have multiple people monitoring the Control - and where all of those people need to receive automated notifications when the Control fails.

*Simple Thresholds will evaluate new points that break their rules whenever the Dashboard’s data view is executed; however, monitoring for these new points is a manual process; hence, the existence of Controls.

Controls Enable Record Keeping
In addition, the Comments created when a Control is broken allow you to keep a record of what your data looked like when failures occurred. When working with dynamic data, using a simple Threshold alone will not necessarily allow you to do this because - depending on Retention settings - newly loaded data can clear out older data and remove points on a Dashboard where Thresholds were broken.

Using a Custom Notification Process
If you would like to create a message of your own, instead of using the system generated message that is typically sent when a control is violated, you can use the Custom Notification Process parameter that is available when building your control. To use this parameter, you will need to set up a Process Model to perform the notification process ahead of time.
Building a Custom Notification Process Model

Below is an example of a typical design for a Custom Notification Process Model to be used with controls.

1. **External Start Event**
   Within this node, you will need to create two String variables, each of the External Property Type: `thresholdResults` and `commentLink`. These are system level variables that will allow you to include data about the Dashboard where the threshold violation occurred in your custom message.

2. **Script Task**
   Within this node, you will need to create Property Reference variables to pass along the `thresholdResults` and `commentLink` variables created in the External Start Event. You should also create any other variables you may need within your script.
   A script such as the following could then be used to process the data associated with your incoming variables.

   ```javascript
   output.thresholdResults = input.thresholdResults;
   output.commentLink = input.commentLink;
   var results = JSON.parse(input.thresholdResults);
   output.dashletId = '"';
   for (var ps in results) {
       output.dashletId += ps;
       output.dashletId += '"';
   }
   ```
3. **Notify Task**

   Within this node, you will once again need to create Property Reference variables for the variables you want to pass over from the preceding Script Task. You may then use those variables in the Notify Task’s Message Details, using the \${variableName} syntax.

   For example:

   ```
   Click here: \${commentLink}
   The dashlet id was: '${dashletId}'
   The threshold rule violation results are: \${thresholdResults}
   ```

*Assigning a Custom Notification Process Model to a Dashboard Control*

Once you have created a Process Model similar to the example described above, you can assign it to the Custom Notification Process parameter in your Dashboard control. After doing so, the next time the Dashboard’s Data View runs, it will use the Process Model and its custom message to notify users, instead of using the standard message.
Part Four: Case Stores

In addition to all functionality previously discussed in this guide, Infogix Data3Sixty also enables Case Work via Case Stores. In Infogix Data3Sixty, Case Work is: 1) the act of defining a process that can push records of interest into a Case Store; 2) the creation of an interface through which case workers can manage cases; and, 3) the actual management of cases by case workers via the interface created in 2.

The content of this chapter is divided into two main sections. Section 1, Building the Case Store Definitions, covers what must be done to accomplish 1 and 2 from the paragraph above. Section 2, Case Management, covers the process of case management by the end users (i.e., “case workers”) of Case Stores.
1. **Building Case Store Definitions**

The first of these two sections, *Building Case Store Definitions*, is provided for users who will be constructing the Analyses that push case data into Case Stores and who will be defining how data within Case Stores is presented to case workers, by creating user interfaces, otherwise known as *Screens*.

Users who will be building Case Store definitions need to be familiar with essentially all other functionality present in Infogix Data3Sixty. They should know how to bring data through all other types of Infogix Data3Sixty Data Stages, as Case Store definition involves many of them.

2. **Case Management**

If you are a case worker, who will be using Infogix Data3Sixty to manage cases, this section of the chapter can help you get started, by showing you how to search for cases, view and edit cases, work with cases that arrive in your Queues, and how to transition cases from one state to another.

---

**Building Case Store Definitions**

*What is a Case Store?*

From the builder’s perspective, a Case Store is a definition that allows you to associate records from other Infogix Data3Sixty Data Stages in a common repository. These records can then be managed by users of the Case Store via screens that you define for the Case Store, to display specific information about cases. The flow of management - that is to say, the series of “states” that a case will go through as it is processed by case workers - is also something that the case store builder must define, via the Case Store’s *Workflow*.

*What is a Case?*

Within the Case Store, an individual “case” is a header record which can be identified by some unique id. A “case” may have a single header record and multiple detail records associated with it that share the same unique id and tie it to the header record; or, a “case” may simply be an individual header record with a unique id and no associated detail records.

*How do I build a Case Store?*

The remaining content of this section first explains the features within each tab you will see on screen while building a Case Store. After functionality is defined, step by step outlines of how to create both a Header/Detail Case Store and a Header Only Case Store are provided.
Case Store Tabs

Here, those responsible for building Case Stores can find basic descriptions of all the features available when building a Case Store. It is recommended that Case Store builders first familiarize themselves with this content, before attempting to create a Case Store using the steps outlined in the following sections. During Case Store creation, this section can then be used as reference material, if questions arise about particular features.

Details Tab
The Details Tab lets you define what type of Case Store you will be creating.

Case contains record(s) from Data Store(s)
This option is what should be chosen when creating a Case Store that will be comprised of both Header and Detail records. “Case contains record(s) from Data Store(s)” means that there is a Data Store of the Internal Database type which contains Detail records, that are associated to the Header records within the Case Store and that can be searched-through and viewed via the Case Store user interface.

Case is not associated with any Record or Stage
This option can be chosen when creating a Case Store that is comprised of only Header records and no Detail records. These Header records may come from any type of Data Store.

Fields Tab
The Fields Tab allows you to view all fields that are built into a Case Store and create/edit new fields to hold the contents of header records pushed to the Case Store.

Built In Fields
Every Case Store comes with the following system fields built in.

CaseId
A number by which to identify case header records within the case store.

CaseUID
A completely unique id that can be used to identify a case header record within a Case Store or between Case Stores. Can also be used to tie detail records from an Internal Database Data Store to cases.

CreatedBy
A code set where values can be ‘System’ (0) or ‘User’ (1).

Status
A code set where values are defined in and synced with the Case Store’s Workflow (described below). Values represent states a case can transition through as it is processed.
**Part Four: Case Stores**

**Building Case Store Definitions**

*Age*
A number representing the age of a case in days.

*Owner*
The name of an Infogix Data3Sixty User who owns the case.

*TransitionedBy*
A code set where values can be ‘System’ (0) or ‘User’ (1). Indicates if the last Transition was performed by the System or a User.

*Type*
A codeset with values defined by the user.

*Priority*
A codeset with values defined by the user.

*QueueId*
The id of the Queue that the case is in.

*RetentionClass*
The Retention Class associated with a record (i.e., a case). Used in the Case Store Other tab to set up retention for individual records. If a record has a null RetentionClass, the record will inherit the Case Store’s Data Retention settings. If a record has a null RetentionClass and no Case Store Data Retention setting exist, the record will stay in the Case Store forever.

*TerminalStateEnteredDate*
The date a case entered the terminal state in the Case Store’s workflow.

**The recCount Field**
When a Case Store of the type *Case contains record(s) from Data Store(s)* is associated to detail record Data Stores in its Workflow, the system will add a new field to the Case Store called recCount (with a number appended to the end*). This field contains numeric values that represent the number of detail records a given case has associated to it.

*The number that is appended to the end of recCount is meant to represent the Associated Data Store for which the counts apply to. This becomes relevant when a Case Store is associated to more than one Data Store.*
Creating New Fields
In addition to the standard fields that are part of every Case Store, the Fields tab allows you to add new fields. This needs to be done for each field that is part of the Header record set that is pushed to the Case Store via Analyses. Giving these fields the same names/data types as the input source fields is not required, although it is recommended.

Creating New Fields via Import
For ease of use, the Import Fields feature can be used to create all of the necessary header record fields from a Data Store, within your Case Store. Imported Fields will have matching names, field types, and all other configurations.

Semantic Type
Applying a Semantic Type to any type of field can help to further categorize the field’s content and enable additional functionality throughout Infogix Data3Sixty.
For example, adding the Airport semantic type to a string field containing airport codes could help to categorize a dataset containing unlabeled airport codes within an Analysis.
Displaying URLs as Hyperlinks in Dashboards with Semantic Type
As another specific example, the URL Semantic Type can be applied to String fields that
contain String values which should be treated as URLs. Doing so will actually cause the
URL field’s Strings to be displayed as clickable hyperlinks in Data Grid dashlets. If the
URLs are valid, clicking on these links will open the associated website in another browser
tab.

Encrypted Fields
Specifying a Case Store field to be encrypted will cause the field’s records to be encrypted
in storage.

Permissions Required to Modify Encryption
- Any user with access to a Case Store can encrypt a Case Store field if it hasn’t yet been
  encrypted.
- Once a field has been encrypted, only Administrators and Users with Administer or
  Manage Security permission to the Case Store will be able to modify the field’s
  encryption settings.

Effects on Data when Changing Field Encryption
- If changing field encryption, a migration job will be performed upon save to encrypt/
  decrypt the Case Store’s pre-existing data.

Secure Fields
Specifying a Case Store field to be a Secure Field will cause the field to be masked
throughout Infogix Data3Sixty. Masking is defined through use of the parameters
described below.

Secure Field Parameters
- **Hide this field when user does not have access**
  Users who have access to a Secure Field are those who belong to the Super Groups
  chosen in the Secure Field’s Super Groups parameter. If this box is checked and a user
  who is not part of the Secure Field’s Super Groups uses the Case Store, they will not see
  the Secure Field at all.
  If this box is left unchecked, all users will see the Secure Field, but its contents will
  always be masked.
- **Mask**
  This setting determines how the Secure Field will be masked.
- **Mask Pattern**
  If the User Defined Mask option is chosen, you can use Mask Pattern to define a preset
  or custom mask for the content of your Secure Field.
Display Mask Pattern
If the User Defined Mask option is chosen, you can use Display Mask Pattern to define
a preset or custom display mask for the content of your Secure Field.

Super Groups
Users in a Secure Field’s selected Super Groups will be able to see the field with its
contents initially masked, as per Mask settings. Unlike other users, however, users
within the selected Super Groups have the option to unmask Secure Fields and see
their actual contents. Unmasking can be performed throughout the product, at the
individual record level or in bulk via Unmask All.
If a user is not in a Secure Field’s selected Super Groups, they will either see the field
as always masked or they won’t see the field at all. This will depend on whether ‘Hide
this field when user does not have access’ has been checked.

Permissions Required to Create or Modify Case Store Secure Fields
- Any user with access to a Case Store can make a Case Store field Secure, if it hasn’t yet
  been marked as secure.
- Once a field has been marked as Secure and the change has been Accepted, only
  Administrators and Users with Administer or Manage Security permission to the Case
  Store will be able to modify the field’s security settings.

Inheritance when used by other Data Stages
- If a field is set as Secure, all the security settings will carry over when the field is used
  in other Data Stages.

Places where you’ll see the effects of Secure Fields
- When you use the Case Store in an Analysis and view its contents in a Sheet
- When viewing a Secure Field within a Case Store Search Screen. Here, you will also be
given the option to unmask the value of a masked Secure Field, if you have been given
permission, by being placed in the Secure Field’s Super Group.
- Dashboards and the Visualizer. Here, some functionality will give you the option to
unmask the value of a masked Secure Field, if you have been given permission by being
placed in the Secure Field’s Super Group.

Code Sets
A Code Set may be used to create a more human readable display value for fields that
contain non-human readable codes. Once a Code Set is created, its effects will be seen
throughout the system - such as when the field’s values are displayed in a Search Screen or
a Dashboard.

Display Name
This is the value that will replace the raw value that is actually stored in the Case Store field,
when the field is displayed throughout the system.
Value
This is the raw value that will be replaced by the display name. Typically, it will be a non-human readable code.

Code Set Example
Assuming a field that contains a set of 3 non-human readable codes, you could create a Code Set for each one, in order to produce a more user-friendly display.

Data Set with Obscure Codes

<table>
<thead>
<tr>
<th>id</th>
<th>obscureCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>GHTY45RD</td>
</tr>
<tr>
<td>002</td>
<td>JJAK09WQ</td>
</tr>
<tr>
<td>003</td>
<td>QWRE21PP</td>
</tr>
</tbody>
</table>

Assuming you created the following:
Display Name: Dental, Value: GHTY45RD
Display Name: Medical, Value: JJAK09WQ
Display Name: Home, Value: QWRE21PP
‘Dental’, ‘Medical’, and ‘Home’ would be displayed for GHTY45RD, JJAK09WQ, and QWRE21PP (respectively) whenever these values were encountered.

Screens Tab
The Screens Tab is where those responsible for creating Case Stores build the UI components that allow case workers to interact with cases. When building a Case Store, two types of screens need to be created: Search Screens and View/Edit Screens.

Search Screens
A Search Screen is what a case worker will use to search for a specific case. As a Case Store builder, you get to define the criteria case workers use to search for specific cases by way of: Filter Fields and Result Fields. You will also need to define how search results appear by specifying a Result View Screen Name and a Result Edit Screen Name.

Filter Fields
Filter Fields are the fields from the Case Store with which you want users to search for cases. Selected fields will appear within the case management interface with the specified Label and Width. Additionally, the chosen Component Type will determine how the case worker can search, where available Component Types depend on the data type of the chosen field.
Wildcard Search for String Filters
String fields used in filters specifically support both literal and wildcard search. Wildcard search can be performed through use of the % character.

Result Fields
Selected Result Fields are those that will appear for header records representing cases returned by a search using Filter Fields. Result Fields will appear as specified by the Label and Width parameters.

Sort Fields
Sort Fields are fields from the Case Store to use for sorting the results of a search. Only fields that are also used as Result Fields may be used as Sort Fields. Additionally, a Sort Field may be specified in Ascending or Descending order, using the Ascending check box.

Result View Screen Name
In order to specify a Result View Screen Name for a Search Screen, you must first have created a View/Edit Screen (detailed below). This selection will determine the display of a case that a user has searched for and then decided to view.

Result Edit Screen Name
In order to specify a Result Edit Screen Name for a Search Screen, you must first have created a View/Edit Screen (detailed below). This selection will determine the display of a case that a user has searched for and then decided to edit.

View/Edit Screens
View/Edit Screens determine how a case is displayed after a user has found it with a Search Screen and then decided to view/edit it. View/Edit Screens essentially allow you to create a form that represents a case, which the case worker may interact with as they manage the case. The form will be comprised of fields from the case, which may be editable or non-editable, based on your requirements. When the case worker works the case, they will use View/Edit Screens to do so.

Screen Layout
When creating a View/Edit Screen, Screen Layout is used to determine the basic arrangement of form fields in the screen.

Field Properties
Once a Screen Layout is created, selecting each field will reveal a Field Properties panel. This panel allows you to define which fields should appear at what points within your layout and how user interaction with the field should occur.
Workflow Tab
The Workflow Tab is where you define the series of States and Queues a case will Transition through as it is processed by case workers. The basic drag-and-drop functionality used to build a Workflow is similar to that found in Infogix Data3Sixty Analyses and Process Models; however, each node and even the connectors between nodes have specific functionality that is unique to Case Stores. This functionality is described below.

Data Store Node
When building Workflows for Case Stores where a Case contains record(s) from Data Store(s), a Data Store node can be used to represent an Internal Database Data Store containing detail records that are associated with the header records in the Case Store. Adding a Data Store node to a Workflow can make it so that whenever the Data Store loads with new detail records, an Analysis that associates those detail records with case header records will execute. The Workflow can also then place newly loaded cases into an initial State and/or Queue.

Associated Data Store
Here you can select a Data Store of the Internal Database type that holds detail records for your cases. Whenever this Data Store loads with new records, the Analysis selected in the Analysis to Execute parameter will run.

Associated Data Store Search Screen
When creating Internal Database Data Stores, you must create a Search Screen for the Data Store. Internal Database Data Store Search Screens are similar to the Case Store Search Screens described above; however, they are what will let case workers search for detail records within an Internal Database Data Store once a header record representing a case has been selected.

This selection must be made here in the Workflow because an Internal Database Data Store may have multiple Search Screens available.

Analysis to Execute
The Analysis to Execute is one that uses the Associated Data Store. It should also be an Analysis that creates new cases and loads them into the Case Store. The idea is that when the Associated Data Store loads with new detail records, the Analysis that builds cases will need to execute. Selecting the appropriate Analysis to Execute automates this execution.
State Node

The State node lets you define what states a case will transition through as it is processed by case workers. With Case Stores, the term “state” is used to refer to a case's status; for example, “Open”, “In Progress”, or “Closed”.

States can be given any name to match the case management requirements of your organization.

Setting a default initial state

You can set a default initial state for all cases that enter a Workflow, which don’t already have a State, by connecting a Data Store node to a State node.

Note that, within each of these analyses, other operations would be required between the input and output nodes. This diagram is primarily intended to show the relationship between a Workflow’s Associated Data Store and the Case Store to which the Workflow belongs.
Part Four: Case Stores

Building Case Store Definitions

Connecting a Data Store node to a State node makes the State the default State for new cases that don’t have a State.

This means that if the Analysis To Execute that is associated with the Data Store runs, and the resulting State field values produced by the run are null, all cases with a State of null will be assigned the default State.

Note: When using this feature, there is no Transition available between the Data Store node and the Default State.

Setting a Terminal State
A State that is marked as a “Terminal State” represents the final state a case will go to when case work is done - for example, “Closed”.

Note: The Fast Track case management feature lets case workers set a case to its terminal state immediately, without having to go through any of its intermediary States.

Transition
Graphically, a Workflow Transition is the connecting line between two Workflow States. When two such nodes are connected, you can click on the connector to display the Transition’s properties panel.

Functionally, the Transition’s properties panel allows you: 1) to create a name for the transition of a case from one state to another, 2) to define which users can perform the transition, 3) to define the screen that case workers will use when performing the transition, and 4) to write a script that will execute when the Transition occurs.

Name
A Transition’s name parameter defines the option that will appear in a case worker’s dropdown menu when transitioning a case to a new state. Any name can be used; however, Transition names should match the State they are transitioning to. For example, when transitioning to the “Assigned” State, you could use a Transition named “Assign”.
Permission
A Transition’s Permission parameter is used to define which case workers can use the Transition. As with other places in Infogix Data3Sixty, permissions are set using Super Groups. If a user is part of the selected Super Group, they will be able to use the Transition to transition the case into a new State.

Edit Screen
A Transition’s Edit Screen parameter lets you create the screen the case worker will use when they perform the Transition. This screen is made available to the case worker to allow them to view/edit fields from the case when it is transitioned.
As the Case Store builder creating this Edit Screen, you should include all of the Fields that are relevant to the case at the point in the Workflow the Transition represents. At a minimum, the case’s Status field should be displayed, since the Transition will affect this field.

Edit Script
The Edit Script feature allows you to write a script that will run when a case worker performs a Transition. A common way to use such a script would be to send an email to a user or a group of users when the Transition takes place, using the sendEmailToUsers or sendEmailToSupergroups functions.

Queue Node
Adding a Queue to a Workflow helps you define which users will be assigned which cases, based on the criteria of your case management process.
For example, you could place all cases that have reached a specific state into a Queue. You could then give a specific group of case workers access to that Queue, so that when they use the case management UI, they see the cases in the Queue and know to work on them.

Name
A Queue’s name parameter defines how the Queue will appear to case workers who have permission to it.

Search Screen
The Search Screen that is selected for a Queue is the Search Screen that case workers will be able to use to search for cases when they are looking at cases within the Queue.

Permission
The Queue Permission parameter defines which Super Groups of users can access the Queue. If a case worker is part of a Super Group that has permission to a Queue, the case worker will see the Queue when managing cases.
Transition To State on Take
When case workers view cases within Queues, they have the ability to Take a case, which will make them the owner of the case. This parameter defines what State the case will transition to when a case worker performs a Take. Available States are all States that occur after the case’s current State, as defined by the Workflow.

Queue Connection
When connecting a Queue node to a State node, you also need to define a Queue Connection. Like a Transition, you can click on a Queue Connection to access its properties.

Name
A Queue Connection’s name parameter simply defines how it appears in the Workflow; this name is not relevant to case workers.

Routing Expression
When creating a Queue Connection, the Routing Expression is what controls whether a case makes it into a Queue.
As an example, consider a scenario where you wanted to define two separate Queues based on each case’s territoryId field value. You could control the flow of cases into these Queues using Queue Connections.

With this setup, each Queue Connector’s Routing Expression would make it so that only cases for Territory 9 went into the Territory 9 Queue and only cases for Territory 10 went into the Territory 10 Queue. As discussed above, access to each Queue could then be restricted to the appropriate case workers, via each Queue node’s Permission parameter.

Basic Workflow Example
Having defined all of the individual components that can comprise a Workflow, we can now provide a basic example which illustrates how the components can be combined.
Description of Basic Workflow Example

1. This Workflow begins with the Internal Database Data Store node. When this Data Store loads, it triggers the execution of another Analysis that loads new records into the Case Store itself.

2. When the Case Store loads, any records that have a null value in their Status field will be assigned the ‘Open’ status. This assignment occurs because there is a State node named ‘Open’ connected to the Internal Database Data Store node.
3. Cases that are in the ‘Open’ State can be transitioned to the ‘Assigned’ State by way of the ‘Assign’ Transition, by those case workers that are part of the Transition’s Super Group. During the Transition, those case workers will also have the ability to edit the case, using the Transition’s Edit Screen. Though it is not shown in the illustration, the Transition could also have a Script, which sends an email to a group leader whenever a case is assigned.

4. Once a case is in the ‘Assigned’ State, a number of actions can occur.
   - If the case satisfies the expression written in the Territory 9 Queue Connection, it will go to the Territory 9 Queue, where it will be accessible to case workers who have permission to that Queue (as defined by the Queue’s Super Group parameter). The same can be said for the Territory 10 Queue. If a case does not belong to Territory 9 or Territory 10, it will go to the All Cases repository, permission to which is set in the Case Store Other tab described below.
   - When a case worker accesses an ‘Assigned’ case, they can either close the case - by using the Close Transition to set its state to ‘Closed’, or they can decide to review the case. If they decide to review, they can use the Review Transition to set the case’s State to ‘Under Review’.
   - After a case has been set to ‘Under Review’, it must then be reopened. Here the case worker will need to use the Reopen Transition to set the case’s State to ‘Open’ once again.
   - If a case is reopened, it can either be Assigned and Closed or it can loop through the Workflow indefinitely.

Indexes Tab
If the search functionality provided in a Case Store’s Search Screen is found to be slow, you can attempt to improve it by creating an Index. Index creation simply involves selecting a field from the Case Store that is used to perform searches.

To gauge whether the Index has improved the Case Store’s search, you should then attempt to search for cases using the same field that was used to create the Index.

Query Filter Tab
Query filters allow you to restrict which data from a Case Store certain users will be able to access. They can be written to apply a Filter Expression to Super Groups of users who query the Case Store and only allow those users to see certain records. When combined with User Profile fields they can also be used to filter queried content at a user level, with no Super Group assignment required. Examples of both types of Query Filter are shown below.
Using a Filter Expression and a Super Group

As an example, consider a data set that contains a state field containing values from all 50 United States (IL, CA, OK, etc.). Also suppose that a system Super Group named ‘Midwest’ already exists, containing users who are only supposed to work with data from the Midwest U.S. region.

To construct a Query Filter that would only allow users in the Midwest Super Group to see data from Midwest states, we could write the following Filter Expression:

\[ state = 'IL' || state = 'IA' || state = 'WI' \]

(Note that more states could be added if they needed to be considered part of the Midwest U.S. region)

We would then need to select the Midwest Super Group as the Super Group to apply this Query Filter to. By default, this Query Filter would then only allow users in the Midwest Super Group to see data from IL, IA, and WI. Importantly, users not in the Midwest Super Group would see nothing.

Allow user to view all data when there are query filters but no matching query filter is found for a user Checkbox

In the section above it is noted that once a Query Filter with a Super Group assignment is applied, the default behavior is that users who are not part of that Super Group will see no data when they query the Case Store. Checking the Allow user to view all data... checkbox will reverse this behavior, allowing users who are not part of the Query Filter’s Super Group to see all data in the Case Store.

Using a Filter Expression with User Profile Fields

As an alternative to applying Query Filters at a Super Group level, Query Filters can be combined with User Profile fields to apply restrictions at an individual user level.

As an example, consider a scenario where the following User Profile fields have already been set up on the following user accounts:

<table>
<thead>
<tr>
<th>User Account</th>
<th>Profile Field Name</th>
<th>Profile Field Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>state</td>
<td>IL,MA</td>
</tr>
<tr>
<td>Jane</td>
<td>state</td>
<td>CA</td>
</tr>
</tbody>
</table>

In this scenario, a Query Filter with the following Filter Expression could be used:

\[ caseStoreStateField = PROFILE_FIELD('state') \]
Here, ‘state’ refers to the Profile Field name set on the user account and caseStoreStateField refers to a state field in the Case Store containing values from all 50 U.S. states. When a user queries the Case Store, the Query Filter will then only show them records where one of their Profile Field values matches the caseStoreStateField value. With this type of set up, no Super Group assignment is required.

Using Multiple Query Filters
When multiple Query Filters are applied, the Query Filters are treated in an OR-like fashion. That is, each Query Filter is considered independently.
For example, if you had the following expressions being used as Query Filters:
1. field1 > 0
2. field2 < 100
Values such as -1 or 101 would both make it through the Query Filters.
To get an AND effect, you would need to combine these expressions into a single Query Filter.

Edit Restriction Tab
Edit Restrictions allow you to restrict which data from a Case Store certain users will be able to edit in Case Management. They apply an Expression to Super Groups of users and only allow those users to edit certain records.

Edit Restriction Example
By default, when no Edit Restrictions are applied to a Case Store, all users with Read Data and Change Data permissions to the Case Store will be able to edit all of its records, using the Edit Screens that have been designed for the Case Store. Adding an Edit Restriction will make it so that users in a selected Super Group have more limited access to what they can edit.

For example, consider a situation where there is a status field in a Case Store, containing values such as Open, Closed, In Review, etc., and that we only want users to be able to edit records that have an Open status. Additionally, suppose that there already exists a Super Group of users named ‘Editors’ to which we would like to apply this restriction.
To do so, we could create an Edit Restriction with the following Expression and apply it to the Editors Super Group:
status = ‘Open’
By default, this Edit Restriction would make it so that whenever users in the Editors Super Group performed Case Management on the Case Store, they would only be able to edit records with an Open status. Users not belonging to the Editors Super Group would not be able to edit anything.
Allow user to edit all data when there are edit filters but no matching edit filter is found for a user Checkbox

In the section above it is noted that once an Edit Restriction with a Super Group assignment is applied, the default behavior is that users who are not part of that Super Group will not be able to edit any records in the Case Store. Checking the Allow user to edit all data... checkbox will reverse this behavior, allowing users who are not part of the Edit Restriction’s Super Group to edit all data in the Case Store.

Using an Edit Restriction with User Profile Fields

Alternatively, User Profile Fields may be used with Edit Restrictions. This type of set up works similarly as in Query Filters, and the example described above for Query Filters could also be applied to an Edit Restriction. The only difference would be that the Profile Field look up would control what records users could edit, not which records they could view.

Other Tab

The Case Store ‘Other’ tab contains miscellaneous functionality that can be used to perform a number of tasks - some of which are required. A basic description of each component found within the Other Tab is provided below.

Dashboard

This setting allows you to associate one or more Dashboards to a Case Store, so that case workers can view visualizations of case data. Any Dashboard may be associated to any Case Store; however, in practice, it will make sense to associate Dashboards with a Case Store that visualize the Case Store directly or that visualize an Internal Database Data Store containing detail records associated to header records in the Case Store.

- Dashboard
  - This is where Dashboard associations are added to or removed from the Case Store.

- Default Dashboard
  - This is the Dashboard that will first appear when case workers use the Case Store.

Case Selection Search Screen

This is the screen that a case worker will use to select a destination case when records are moved between cases, using the Move feature on detail records or the Merge feature on header records.

My Cases Search Screen

When a user has been assigned the role of ‘Case Worker’ by your Infogix Data3Sixty Administrator, they will not be able to see all of the Search Screens you have created for a Case Store.

Setting the My Cases Search Screen defines what Search Screen case workers will be able to use to search for cases within their My Cases repository.
All Cases Search Screen
When a user has been assigned the role of ‘Case Worker’ by your Infogix Data3Sixty Administrator, they will not be able to see all of the Search Screens you have created for a Case Store.

Setting the All Cases Search Screen defines what Search Screen case workers will be able to use to search for cases within the All Cases repository, provided they have access to it.

Super Groups with Access to All Cases
Within the Case Management UI, there are two repositories for cases: My Cases and All Cases. Users within the Super Groups selected for this parameter and users who are Infogix Data3Sixty Administrators will be the only users that will be able to access All Cases.

Data Retention
This property is used to set a time limit on how long a Case Store will retain incoming data. It is useful in cases where data becomes irrelevant after a certain amount of time has passed.

Retention Period defines the length of time for which a Case Store will store data. Data Retention applied at the Case Store level will override Retention Period settings at the Environment level made by your Administrator. Time is always calculated from the time data is loaded into a Data Stage.

Period Precision defines the precision at which a Retention Period is applied.
For example, if a Case Store’s Retention Period was set to 1 year and its Period Precision was set to Month, the Retention Period would span one year back from the current day, plus the remaining days to get to the beginning of the month.

Terminal State - Importantly, Data Retention is only applied to cases that have reached the Terminal State in their Workflow.

Retention Example 1

For instance, suppose the current day is 6/5/2016.
If Case Store Retention Period was set to 1 year and Period Precision was set to Month, the system would first look back one year to 6/5/2015. Then, since Precision is set to Month, the system would go to the beginning of the month, to 6/1/2015. At this point, any case that was loaded into the Case Store before 6/1/2015 and had reached the Terminal State in its workflow would be removed.

Retention Example 2

As another example, suppose the current day is 6/5/2016. If Environment Retention Period was set to 1 year and Period Precision was set to Quarter, the system would first look back one year to 6/5/2015. Then, since Precision is set to Quarter, the system would go to the beginning of the quarter, to 4/1/2015. At this point, any case that was loaded into the Case Store before 4/1/2015 and had reached the Terminal State in its workflow would be removed.

Retention Period Over Time
Data Retention is applied once per day, and is calculated from the end of the most recent successful load. If a Run or Rebuild fails, Data Retention will be calculated from the failed Run/Rebuild’s start time.
Retention Period Data Flush

Over time, data will be flushed from Case Stores based on the Data Retention settings specified and whether a case has reached the Terminal State in its workflow. For instance, in the example above, cases from Load 1 on 4/29/2015 that have reached their terminal state are flushed on 5/1/2016. This is because a 1 year retention period with Month precision is used; so, on 5/1/2016, the 4/29/2015 Load falls outside of the calculated Retention period (which spans from 5/1/2015 to 5/1/2016).
Alternate Retention Policies
In addition to basic Data Retention functionality, Case Stores also feature the ability to create alternate retention policies, using a retention class.

To do so, you’ll need take advantage of the RetentionClass field that the system automatically assigns to all Case Stores. By default, records in this field are assigned a value of null, which will make it so that, by default, all records in the Case Store will use the Data Retention settings that have been applied at the Case Store level. If you populate the RetentionClass field with Strings within the Analysis, however, you can use these same Strings as Retention Classes within the Other tab and define unique retention policies for each one.

Alternate Retention Policy Example
For example, if you wanted to create two separate retention policies based on criteria from the fields that comprise a record, you could create a new column at some node preceding the Case Store Output of an Analysis, using an expression that controlled the field’s value.

1. Within the Select node, create New Column:
   IF(claimAmount >= 3000, 'RetentionClassA', 'RetentionClassB')

2. Within the Case Store Output node, map New Column from Select node to Case Store Output’s RetentionClass field.

Here, as an example, a Select Node precedes a Case Store Output node. Within the Select node, we can create a new column using the expression: IF(claimAmount >= 3000, 'RetentionClassA', 'RetentionClassB'). Within the Case Store Output node, we can then map the new column to the RetentionClass field.

After setting up our Analysis, we can then go into the Case Store’s Other Tab and create an Alternate Retention Policy for each newly added class. This would require giving one policy a Retention Class named ‘RetentionClassA’ and giving the other policy a Retention Class named ‘RetentionPolicyB’. Each policy could have unique Period, Precision, and Period Precision parameters.
With Alternate Retention Policies set in place, system retention processing would now treat records within the Case Store individually. Records that did not fall into an alternate class would have a default RetentionClass value of null and thus inherit the Retention settings of the Case Store.

Additional Case Store Retention Processing Tasks
In addition to the Case Store itself, retention processing will also delete records from Data Stores owned by the Case Store, if there are Data Store records that are associated with a case that is being deleted by retention processing. This relationship is determined via the Data Stores’ OwningSubObjectId field.

Background Processing
Within the Other tab, you may also create a Script. A Case Store’s Script will be run whenever the Case Store is Executed. Executions may be performed manually or they can be scheduled.

Background Processing Script Syntax
Throughout the product, Script syntax follows the general format of referencing inputs to the Script with the input keyword and outputs of the Script with the output keyword. This syntax differs for Case Store Background Processing.

To reference any Case Store field in your script, regardless of whether it is an input or an output, use the caseData keyword.

For example:

```java
if(caseData.Age <= 30) {
    caseData.Age_Bucket = ‘Under 30’;
}
```

The above Script would check each record’s system field Age, to evaluate whether its value was less than or equal to 30. If it was, it would assign a value of ‘Under 30’ to the record’s Age_Bucket field.

Note that as with assigning output values to any new field in a Script, a prerequisite to running this script would be creating the Age_Bucket field within the Case Store definition.

Building a Case Store for a Header/Detail Relationship:
The ‘Case contains record(s) from Data Store(s)’ Type
In this section, we will present the steps required to create a Case Store of the type: Case contains record(s) from Data Store(s). This type of Case Store is intended to support a Header/Detail relationship, where header records representing individual cases reside in the Case Store and detail records that support individual cases reside in an associated Internal Database Data Store.
In essence, this requires: 1) Pushing header records to a Case Store via Analysis, 2) Pushing detail records to an Internal Database Data Store via an Analysis, and 3) Defining a Case Store so that case workers can view header and detail records together in one place.

Steps to Configure a ‘Case contains record(s) from Data Store(s)’ Case Store

1. **Configure an Internal Database Data Store for Detail Records**
   Due to dependencies, you need to create a Data Store of the Internal Database type first. This Data Store should have:
   - All the fields your detail records will need. The fields can be created within the Data Store Fields tab using Import Fields; or, if you are creating the Data Store on the fly in an Analysis they will be created automatically in the Data Store Output node’s properties panel.
     Alternatively, a Case Output Internal Database Data Store from Infogix Assure can be used - in which case, Infogix Assure will be populating the Data Store and Step 2 can be skipped.
   - A Search Screen, to integrate the Data Store into the Case Store. This Search Screen is what will be used to find detail records from the Data Store while working within the Case Store. This search screen must be created in the Data Store’s Screens tab.
   - A set of Identity fields, with Identity fields are unique set to True. This configuration must be set in the Data Store’s Identity tab.

2. **Build an Analysis that pushes detail records to your Internal Database Data Store. Do not execute it yet.**
   The Analysis should be one that somehow finds detail records that need to be turned into cases.

3. **Create the Case Store.**
   - Within the Details tab, choose Case contains record(s) from Data Store(s) as Case Type.
   - Within the Fields tab, create appropriate fields for the header records that will be pushed to the Case Store. Use of the Import Fields feature is recommended for this task.
   - Within the Screens tab, create at least one Search Screen and one View/Edit Screen. The Search Screen is what will be used by case workers to search for header records (i.e., cases) within the Case Store. The View/Edit Screen defines how cases returned by search will be displayed.
   - Save and Exit the Case Store.

4. **Build a Case Maker Analysis.**
This Analysis should associate header and detail records within a Co-Group Node by creating a new column using the UUID() function.

It should then push header records to the Case Store and detail records to the Internal Database Data Store.

- The UID from the Flattened Detail node should be mapped to the OwningSubObjectUID field in the Internal Database Data Store. This field will not exist in the Internal Database Data Store, however, until you complete Step 5, because it is automatically added to the Internal Database Data Store when the Data Store is associated to the Case Store in its Workflow. In practice, you will need to save your Case Maker Analysis, add a Workflow containing an Associated Data Store to your Case Store, and then return to your Case Maker Analysis to make this mapping.

- The UID from the Flattened Header should be mapped to the CaseUID field in the Case Store. All Case Stores have a CaseUID field for this purpose.

- If data from pre-existing cases is being loaded, a new UID should not be assigned and the update functionality in the Data Store and Case Store Outputs should be used.

5. **Return to the Case Store, and create a Workflow.**

- Within the Workflow tab, create the Workflow for cases in the Case Store.
Drag in a Data Store node. This will be used to orchestrate execution of the Analysis that loads the Case Store with header records with the loading of the Internal Database detail record Data Store.

To associate the Case Store with the detail record Data Store created in Step 1, include a Data Store node in your Workflow and select the detail record Data Store as the Associated Data Store within the node’s properties.

To orchestrate the Case Store with the loading of the detail record Data Store that has been associated with the Case Store, use the Data Store node’s Analysis To Execute property. The selected Analysis should be one that loads the Case Store with header records. Once the Analysis is chosen as the Analysis To Execute, it will execute automatically whenever the Associated Data Store loads new data.

6. Return to the Case Maker Analysis and Map the UID field from your Detail Records to the OwningSubObjectUID field in the Internal Database Data Store Output node.

7. Execute the Analysis that loads the Detail Record Store.
At this point, you will have performed all steps necessary to associate a detail record Data Store with header records in a Case Store. You can now execute the Analysis that loads the detail record Data Store. Once the detail record Data Store loads, execution of the Analysis that loads the Case Store with header records will be triggered.

**How to Check if Records already belong to a Case**

In the basic example shown above, a “Case Maker” Analysis is shown assigning UIDs to new header and detail records that comprise cases. This basic example does not, however, include any logic to check whether incoming header records already have a CaseUID or incoming detail records are already associated to a case via an `OwningSubObjectUID` value. The following Analysis example shows one way that this logic can be implemented.

1. **Assign New Records To Existing Cases**

   In this first step, a Join is performed using a common field from each input source (here we will call this common field `OrderId`). This Join allows us to associate header records from our Header Data Store and detail records from our Detail Data Store. To represent this relationship with a field value, we then create a New Column called `NewOwningSubObjectUID` in the Join node using the following expression:

   \[
   \text{IF}(!\text{ISNULL}(\text{OwningSubObjectUID}), \text{OwningSubObjectUID}, \text{IF}(!\text{ISNULL}(\text{CaseUID}), \text{CaseUID}))
   \]

   `OwningSubObjectUID` is a detail record field that holds the UID values of case header records that own detail records. This expression will therefore: check to see whether a detail record has an `OwningSubObjectUID`; if it does, it will keep and use that value for `NewOwningSubObjectUID`; if it does not, it will check to see whether the header record has a `CaseUID`. If the header record does have a `CaseUID`, then it is a
preexisting Case and the detail record should use that CaseUID as the value of NewOwningSubObjectUID. If, on the other hand, the header record does not have a CaseUID, then it is a new Case entirely - meaning it will need to be assigned a new CaseUID further on in the Analysis and that the detail record NewOwningSubObjectUID field should not be assigned any value yet.

2. **Perform Filtering to Separate Preexisting and New Records**

In this next step, two Filter nodes are used to separate preexisting header records from new header records that need a new CaseUID.

**New Sales Order Filter (top)**

This filter uses the following expression, to find preexisting header records that have new detail records that should be associated to them:

\[
\text{ISNULL(OwningObjectId)} \&\& (\text{ISNULL(SysDeleted)} || (!\text{SysDeleted}))
\]

This expression finds records that do not have a value for the OwningObjectId field yet - meaning new detail records that have been associated to header records via a NewOwningSubObjectUID value but that have not yet been part of the Case Store (otherwise they would have a non-null OwningObjectId).

Additionally, we will know that these are new detail records because they have not been assigned a SysDeleted value yet or have a SysDeleted value of False.

**New Sales Order w/out Cases Filter (bottom)**

This filter uses the following expression, to find entirely new header records that will need a new CaseUID:

\[
\text{ISNULL(NewOwningSubObjectUID)} \&\& \text{ISNULL(OwningObjectId)}
\]

This expression finds detail records that have been associated to header records via a common OrderID but have not yet been assigned a value for NewOwningSubObjectUID.

Here, we know that any header and detail records that enter the filter are entirely new to the Case Store because the expression used to assign a value to NewOwningSubObjectUID in Step 1 returned null and also because OwningObjectId is null.

3. **Assign new CaseUIDs to new cases**

Once we have distinguished preexisting cases from new cases, we can find distinct records using the OrderId field in a Distinct node. Having identified distinct, new cases, we can then assign each case a new CaseUID by creating a New Column called NewCaseUID and simply using the UUID() function to populate this field’s values.
4. **Assign a FinalOwningSubObjectUID**
   At this point we can once again perform a Join using the field that is common to both header and detail record data sets, *OrderId*. We can then create a New Column using the following expression, in order to create a *FinalOwningSubObjectUID* value for detail records:

   ```plaintext
   IF(!ISNULL(NewOwningSubObjectUID), NewOwningSubObjectUID, NewCaseUID)
   ```

   Effectively, this expression assigns the *NewCaseUID* that was created in Step 3 to new detail records that should be associated to new cases. It will also retain any non-null *NewOwningSubObjectUID* values that were created in Step 1, assigning these values to detail records that are associated to preexisting cases.

5. **Isolate Detail Record fields with a Select node**
   Now that all detail records have a *FinalOwningSubObjectUID* value, we can isolate all of our fields that belong to detail records so that we can eventually push them to our Data Store Output in Step 8.

6. **Isolate Header Record fields with a Group By node**
   Similarly to Step 5, we also need to isolate header record fields that should be pushed to our Case Store Output. This is done by grouping by *OrderId* with a Group By node, and then creating a New Column for each header record field. Since we have used a Group By node, these header record fields must be populated with Aggregation functions; however, since each *OrderId* group should only have one record within it, aggregation will effectively pass a single value. For example, to pass our *FinalOwningSubObjectUID* created in Step 4, we can create a New Column in the Group By node called *CaseUID*, using the following aggregate expression:

   ```plaintext
   MAX(FinalOwningSubObjectUID)
   ```

   Similarly, we can use the MAX() function to generate values for each field that needs to be pushed to the Case Store Output from our header records.

7. **Map to Outputs and Set to Update**
   Once we have isolated the detail fields that need to be pushed to our detail Data Store and the header fields that need to be pushed to our Case Store in Steps 5 and 6, we can map to these fields in their respective output nodes. Since some records may be preexisting, we also need to set our outputs to Update on a key field, so that preexisting records are updated rather than added again and duplicated.
Building a Case Store for a Header Records Only: The ‘Case is not associated with any Record or Stage’ Type

In this section, we will present the steps required to create a Case Store of the type: Case is not associated with any Record or Stage. This type of Case Store is intended to support header records only, where header records representing individual cases reside in the Case Store.

1. **Create the Case Store.**
   - Within the *Details* tab, choose *Case is not associated with any Record or Stage* as Case Type.
   - Within the *Fields* tab, create appropriate fields for the header records that will be pushed to the Case Store. Use of the *Import Fields* feature is recommended for this task.
   - Within the *Screens* tab, create at least one Search Screen and one View/Edit Screen. The Search Screen is what will be used by case workers to search for header records (i.e., cases) within the Case Store. The View/Edit Screen defines how cases returned by search will be displayed.
   - Within the *Workflow* tab, create a Workflow. The Workflow may include State nodes and Queue nodes. It should not include any Data Store nodes, as Internal Database Data Stores are not used with this type of Case Store.

2. **Build an Analysis that pushes header record data to your Case Store.**
   Important: For this type of Case Store, the Analysis will need to include the creation of a new String type field with a value of ‘wf-state’ for every record. This newly created field should be mapped to the Case Store Output’s Status field. Doing so will set the initial Status of cases within the Case Stores to what is effectively a null value.
   Also note that this step is not required when working with Case Stores of the type *Case contains record(s) from Data Store(s)*, as it happens automatically.

At this point, you will have performed all steps necessary to load a Case Store with header records. You can now execute the Analysis that loads the Case Store.

**Case Management**

This section of the guide focuses on how to work with Case Stores that have been built for Case Management. The section assumes that a Case Store has already been properly constructed for use in your system, but it does not require that you were the one who constructed it. Knowledge of how to build a Case Store is not strictly required to perform case work; however, it can help.

Note: This section may also be useful for those building Case Stores, as it can help to inform how
what they’re building will appear to end users.

**Accessing Case Stores**

Case Stores that you have access to will be available via Infogix Data3Sixty’s Main Menu, listed within the *Manage Cases* option. When you click on a listed Case Store, you will be brought to the series of screens that have been built for you to work cases.

**Case Manager and Case Worker Roles**

Before diving into the details of each screen available for use within the Case Management UI, it’s important to note the existence of security roles and how these will affect what you can see.

*Case Manager*

Case Managers have more rights than Case Workers. If you are a Case Manager, you will see all components of the Case Management UI.

*Case Workers*

Case Workers have a more limited view of the Case Management UI. In particular, they will not see all of the Search Screen options that Case Managers see.

**My Cases**

The *My Cases* repository is meant to contain every case within a Case Store that is owned by the current user.

*Searching My Cases*

When you have clicked My Cases and are not inside of a Queue, you will see all of the cases you currently own. The Case Store will also have a default Search screen configured that allows you to search through these cases.

If your account has been assigned the Case Worker role, this default Search screen will be your only option for navigation through the *My Cases* repository. If, on the other hand, you’re a Case Manager, you’ll have the ability to use different Search screens to navigate My Cases via the Search section described below.

**All Cases**

The *All Cases* repository contains every case in the Case Store, including those that have been pushed to your Queue(s) and those that have not. Depending on your user permissions, you may or may not have access to All Cases.
15 ■ Part Four: Case Stores

Case Management

Searching All Cases
When you have clicked All Cases and are not inside of a Queue, you will see all of the cases in the Case Store. The Case Store will also have a default Search screen configured that allows you to search through these cases.

If your account has been assigned the Case Worker role, this default Search screen will be your only option for navigation through the My Cases repository. If, on the other hand, you’re a Case Manager, you’ll have the ability to use different Search screens to navigate All Cases via the Search section described below.

Dashboards
Dashboards that you find within the Dashboards section are those that are relevant to the cases within the Case Store. In most situations, they will be visualizations of the records that comprise the Case Store (either the header records from the Case Store itself or detail records from an Internal Database Data Store that is associated with the Case Store).

Queues
Cases will go into Queues based on the Case Store’s Workflow, a component created by the person who created the Case Store.

From the perspective of a Case Manager or Case Worker, the Queues listed in the Queues section are the Queues they have permission to; i.e., other users may have permission to other Queues that you don’t see when you log into your account.

If a case is in your Queue, it has been placed there because you are the person who should be working on it. Details on how to ‘work’ a case are provided in a later section of this chapter.

Searching Queues
When navigating your Queues, you will have the ability to search through them using predefined Search screens configured by the person who built the Case Store. In the context of a Queue, this ability to search appears on the top half of the screen, and results of the search appear on the bottom.

Search (Case Managers Only)
Users who have been assigned the Case Manager Role will also have a Search option, under Queues. The Search Screens listed in the Search section are all of the Search screens that have been created for the Case Store. Search Screens may be used to search My Cases and All Cases.
Working Cases

When using the Case Management sections outlined above, you have the ability to locate specific cases via Search. Once a specific case is found, you can click on its record and a number of features become available for processing the case. The following section provides an overview of these features.

Viewing a Case

When a case is selected and the View button is used, you will be brought to a predefined View Screen showing you fields from the case. In View, field values are not editable.

Note to Case Store creators: The View screen users see after clicking View on a case is the one that was selected as the Result View Screen name within the definition of the Search Screen the user used to locate the case.

Editing a Case

When a case is selected and the Edit button is used, you will be brought to a predefined Edit Screen showing you fields from the case. In Edit, field values are editable. To date, inline editing is also supported, meaning that you may also simply double click a selected record and change its contents inline, without have to go to its Edit screen.

Note to Case Store creators: The Edit screen users see after clicking Edit on a case is the one that was selected as the Result Edit Screen name within the definition of the Search Screen the user used to locate the case.

Mass Editing Cases

If you would like to edit multiple records at once, you can multi-select the records using CTRL-click. Once you have done so, you can then utilize the Mass Edit button, to edit all of the selected records at once. When using Mass Edit, all fields that are available to edit will appear in the form; however, only those fields on which you have checked Include will have the form’s changes applied to the selected records.

Deleting a Case

If a case needs to be deleted from the Case Store, it can be selected and the Delete button can be used. Note that multi-select-and-delete is supported as well.
Transitioning a Case

When it comes time to move a case from one Status to another, the Transition dropdown should be used. The Transition options available in this dropdown are those that map to specific states that a case needs to go through as it is processed. These states will have been predefined by whoever created the Case Store, and they will likely emulate a workflow that you use within your organization.

Once a case has been transitioned, the value within its Status field will change accordingly.

Quick Guide to Transitioning a Case

Transition Menu Item Visibility
The Transition Menu will not show menu items unless they have been defined in the Case Store’s Workflow.

Enabling a Transition Menu Item
A Transition menu item will only be enabled when all selected cases to transition have the same State.

When are Transition screens shown?
If a Transition menu item is chosen, a Transition screen will only be shown if a screen has been defined in the Case Store’s Workflow for the chosen Transition. Also note that if only 1 case is chosen to Transition, the Transition screen will be populated with that case’s data; when multiple cases are selected, this form population does not occur.

What happens to case data collected on a Transition screen?
All case data collected on a Transition screen will be applied to all the cases you have selected to Transition.

What is the resulting state of cases after using a Transition?
After selecting a Transition and using its Transition screen, all selected cases will transition to the selected Status.

What is the resulting Queue Id of a case after using a Transition?
After selecting a Transition and using its Transition screen, Queue Ids of the selected cases will not be affected unless the cases were explicitly assigned to another user in the Transition screen. If cases have been assigned to another user, their Queue Ids will be cleared.

What is the resulting Owner of a case after using a Transition?
After selecting a Transition and using its Transition screen, Owners of the selected cases will not be affected unless the cases were explicitly assigned to another user in the Transition screen.
Transition Script Processing
If a Transition script from the cases’ current state to the ‘To’ state is defined, then this script will be evaluated and change the data of each case. Note however that if a Transition script modifies any System Fields, these changes will be lost.

Evaluation of Queue Expressions
When transitioning with a Transition screen, if a case’s Owner is explicitly changed, then Queue Expressions are not evaluated. Otherwise, if there are Queues associated with the transitioned to State, the Queue Expressions will be evaluated and the first one that returns true for a given case will determine the Queue Id. If a Queue Id is determined for a case, then its Owner field will be cleared. If, on the other hand, none of the queue expressions return true, neither Queue Id nor Owner will be changed.

Terminal State Processing
If a case is transitioned to a Terminal State, the case Owner will be cleared.

Merging a Case
If for some reason you find that two or more cases actually represent the same case, you can use the Merge Cases feature to combine them into a single case. To do so, multi-select the duplicate cases and the Merge Cases button will become enabled.

Selecting a Target Case
When using the Merge Cases feature, you will also need to select a “Target Case”. This is be the case that the other cases will merge into. After merging the cases, the Target Case will remain and the cases that were merged will be moved to the Case Store’s Terminal State.

Taking a Case
Taking a Case will change the value of the case’s Owner field to your name. If the case has been taken from within a Queue, Take may also change the State of the case, based on the Case Store’s Workflow.

Quick Guide to Taking a Case
Enabling the Take Button
To enable the Take button, all selected cases must be in the same Queue and this Queue must be one that the current user has access to.

When are Transition screens shown?
If the Take button is used, a Transition screen will be shown if the current Queue has had a Transition to State On Take defined in its corresponding node within the Case Store’s Workflow and a screen has been defined in the Transition to that State in the Case Store’s
Working Cases

Workflow. Also note that if a Transition screen is shown after clicking the Take button, any Owner field shown in a Transition to screen will be disabled, to prevent changing the meaning of “Take”.

What happens to case data collected on a Transition screen? During a Take, all case data collected on a Transition screen will be applied to all the cases you have selected to Take.

What is the resulting state of cases after using Take? If the Queue node within the Case Store’s Workflow that corresponds to the Queue you are currently in has had its Transition to State On Take parameter configured, all selected cases will transition to that Transition to State On Take.

What is the resulting Queue Id of a case after using Take? After Taking a case, its Queue Id is cleared.

What is the resulting Owner of a case after using Take? After taking a case, you - the current user who took the case - will become its Owner.

Transition Script Processing
If Transition To State on Take has been set, Transition scripts will be evaluated and change the data of each case that has been taken. If Transition To State on Take has not been set, Transition scripts will not be evaluated. Note however that if a transition script modifies any System Fields, these changes will be lost.

Evaluation of Queue Expressions
Queue Expressions are not evaluated when taking a case.

Terminal State Processing
If a case is transitioned to a Terminal State during a Take, the case Owner will be cleared.

Fast Tracking a Case
Instead of Transitioning a case through its multiple states, you also have the option to Fast Track the case to its terminal state. Doing so will simply advance the case to the last State in its Workflow, which will likely be something like ‘Closed’.

Quick Guide to Fast Tracking a Case

Fast Track Button Visibility
The Fast Track button becomes visible when at least one State within the Case Store’s Workflow is marked as Terminal.
Enabling the Fast Track Button
The Fast Track button will not be enabled unless the current user has Admin permission to the Case Store.

When are Transition screens shown?
Transition screens are not shown when using the Fast Track button.

What is the resulting state of cases after using Fast Track?
Fast Tracking a case takes it directly to the Terminal state, defined in the Case Store’s Workflow.

What is the resulting Queue Id of a case after using Fast Track?
After Fast Tracking a case, its Queue Id is cleared.

What is the resulting Owner of a case after using Fast Track?
Fast Tracking a case will clear the case’s Owner field.

Transition Script Processing
Transition scripts are not evaluated when Fast Tracking a case.

Evaluation of Queue Expressions
Queue Expressions are not evaluated when Fast Tracking a case.

Unmasking Secure Fields in Case Stores
If you have permission to unmask a secure field, you may unmask it at the record level using the lock icon that appears in records or you may unmask records in bulk using the Unmask All button. Note that every time you perform unmasking, your action will be logged in the product’s Audit Trail, accessible to administrators.

Detail Records from Associated Data Stores
If working with a Case Store of the type Case contains record(s) from Data Store(s), Viewing a Case will also display detail records that are associated to the case in the bottom half of the screen. These records may also be worked with, using a feature set similar to that which is available when working with header records.

Records Tab
When Viewing a case, Detail Records from Data Stores that are associated to a Case Store are displayed within the Records tab. Like header records that represent the case, detail records may also be Viewed and Edited. Unlike header records, detail records have an additional feature called Move.

Note to Case Store creators: The search Filter used to search for detail records and the result fields that are returned are defined by the Associated Data Store Search Screen parameter in the
associated Data Store node that was used in the Case Store Workflow.

**Viewing Detail Records**
When a detail record is selected and the View button is used, you will be brought to a predefined View Screen showing you fields from the detail record. In View, field values are not editable.

Note to Case Store creators: The View screen users see after clicking View on a detail record is the one that was selected as the Result View Screen Name parameter in the associated Data Store’s Search Screen that was selected in the Case Store Workflow.

**Detail Record History**
When viewing individual detail records, a history of record activity is also made available. This functionality shows a listing of all changes that have occurred to the detail record via its Edit screen.

**Editing Detail Records**
When a detail record is selected and the Edit button is used, you will be brought to a predefined Edit Screen showing you fields from the detail record. In Edit, field values are editable, and when they are edited the change will be tracked within the Detail Record History that is accessible via View.

Note to Case Store creators: The Edit screen users see after clicking Edit on a detail record is the one that was selected as the Result Edit Screen Name parameter in the associated Data Store’s Search Screen that was selected in the Case Store Workflow.

**Moving Detail Records**
When a detail record is selected, it may also be Moved to a different case entirely. Doing so will associate it with a new header record and remove the association to the original header record.

**Notes tab**
When a case is selected, you have the ability to make notes about it within the Notes tab. Notes functionality works similarly to the Comment functionality found elsewhere throughout Infogix Data3Sixty, in that you may Add, Edit, Reply, and view your notes in Thread form.

**Attachments tab**
External files that are related to a case may be attached to it while it is being viewed, via the Attachments tab.

**History tab**
Similar to the History tab that is available at the detail record level, case header records also have such a tab. Events that are logged here pertain to the case header record as a whole.
Data Entry

If you have been given permission to perform Data Entry and you have Read Data and Change Data permissions to at least 1 Internal Database Data Store, you can modify its records through the Infogix Data3Sixty UI. This functionality is found within the product’s main menu, under Data Entry. Once Data Entry is selected, you will be able to select an Internal Database Data Store to search or modify.

Searching the Data Store

When you have selected a Data Store to perform Data Entry on, you will be brought to a screen containing all of its Search screens. These Search screens will have been built by someone within your organization when the Internal Database Data Store was defined. The screens will contain Filter Fields, with which you can perform your search, and Result Fields, with which you will see the results of your search. Additionally, the results contained within the Result Fields will appear sorted, according to the Sort Fields that have been specified for the Data Store.

When you have chosen a Search Screen, you can use the top panel containing Filter Fields to perform your search. The results of your search will be displayed in the bottom panel, as defined by the Data Store’s Result Fields and Sort Fields.

Viewing Records

When search results are returned in the bottom panel, you will have the option to View a record by selecting it and using the View button. Doing so will bring up the screen that has been defined as the Search Screen’s Result View Screen in the Data Store definition. This screen will allow you to see the contents of whatever fields have been included in the Result View Screen. It will also contain a History of the record’s activity, such as when it was created and if it has been edited.
**Editing Records**

When search results are returned in the bottom panel, you will have the option to *Edit* a record by selecting it and using the Edit button. Doing so will bring up the screen that has been defined as the Search Screen’s *Result Edit Screen* in the Data Store definition. This screen will allow you to edit the contents of whatever fields have been included in the Result Edit Screen.

*Inline Editing with Double Click*

If a Result field has been defined as Editable, you can double click its cell’s value to perform inline editing. This will enable you to quickly change records without having to use the record’s Edit screen.

*Edit Restrictions*

When an Internal Database Data Store is defined, an Edit Restriction may be applied to it. The Edit Restriction can be used to prevent certain users from editing certain records. If you find that there are records that you are unable to edit, it may be because an Edit Restriction has been applied. To find out if this is the case, you will need to consult the person who defined the Data Store you are working with.

**Deleting Records**

When search results are returned in the bottom panel, you will have the option to *Delete* a record by selecting it and using the Delete button. Doing so will remove the record from the Data Store.

**Creating Records**

When working within Data Entry, you may also use the bottom panel to add records to a Data Store. This is performed by using the Create button. When adding records via the Create button, you will use a form containing fields that has been predefined by the person in your organization who created the Data Store definition. Within the Data Store’s Search Screen definition, this is referred to as the *Create Screen Name*.

**Unmasking Secure Fields**

If you have permission to unmask a secure field, you may unmask it at the record level using the lock icon that appears in records or you may unmask records in bulk using the Unmask All button. Note that every time you perform unmasking, your action will be logged in the product’s Audit Trail, accessible to administrators.
General Product Functionality

This section of the User Guide covers features found throughout the product that do not apply to any one Data Stage in particular.

Using Pipelines Across Multiple Instances

For organizations using Infogix Data3Sixty across multiple instances, Infogix Data3Sixty also includes Import/Export Definitions functionality, to move Data Stages into and out of the platform. This functionality is available by way of the More button; however, for non-admin users, only Export is available for use. A more detailed description of Import is available in the Infogix Data3Sixty Administrator Guide.

Exporting Definitions

What am I Exporting?
When Exporting, you are exporting your Data Stage Definitions - i.e., the rules that make them what they are, behind the scenes of Infogix Data3Sixty. You are not exporting any data associated with them.

For example, if you export a Data Store and then Import it to another instance, the Data Store will not contain any data; it will, however, contain all of the fields and settings you applied to it in its original instance.

This means that after importing to a new instance, you will need to reload your Data Stages with the relevant data.

What Should I Export?
You can export any individual Data Stage Definition you’d like.

For simplicity, however, it is highly recommended that you export entire Pipelines. This is because to properly Import a Data Stage Definition, the destination instance needs to contain the parents of the Data Stage (its Pipeline and Path) and any dependencies the Data Stage may have.
For example, if you simply exported an Analysis Definition and then attempted to import it to another instance, the Import would fail for two reasons. The first reason would be that the Analysis’ parent Pipeline and Path don’t exist in the new instance. The second reason would be that the Data Stores the Analysis uses don’t exist in the new instance either.

To remedy this, you could simply bundle the Analysis and its Data Stores into a common Pipeline, and then export the entire Pipeline Definition.

**How do I Export Definitions?**

To Export a Data Stage Definition, simply select it within the Pipeline context and use the Export Definitions option available in the More dropdown. Doing so will initiate the download of a zip file. This zip file is what you import to your new instance.

**Dirty State when Exporting Definitions**

When working with Pipelines, you may notice that some items have an asterisk to the right of their name. This marking signifies the item’s ‘dirty state’, which is a flag that exists to help you track whether the item’s definition has changed since the last time it was promoted or exported.

After creation, all new Data Stages will have a dirty state of true, and will be marked with an asterisk next to their name. If the Data Stage Definition is exported to another environment, you will have the option to reset its dirty state. If you reset, dirty state will be set to false and the asterisk will disappear. If after this reset the Data Stage changes, dirty state will again be set to true and the asterisk will reappear.

**Promoting Pipelines Across Multiple Environments**

For organizations using Infogix Data3Sixty across multiple environments, the product also includes Promote functionality, to move Data Stages from one Environment to another. This functionality is available to admin users by way of the More button. A more detailed description of Promote is available in the Infogix Data3Sixty Administrator Guide.

**Navigating to a Selected Stage**

Throughout the product, functionality exists to “Go to a selected stage”. This functionality is a way to quickly move from one Data Stage to another, and it is available in places where one Data Stage uses or references another. When clicking “Go to Selected Stage” you will simply be transitioned to the View mode of the Data Stage that you have selected.
The Go to Selected Stage feature is currently available when:

- Viewing or Editing an Analysis, when highlighting a node that represents another Data Stage; for example, when highlighting a Data Store Input node.
- Viewing or Editing a Process Model, when highlighting a node that represents another Data Stage; for example, when highlighting an Execute Stage Task.
- Viewing the Executions History of an Analysis or a Process Model, when highlighting a node that represents another Data Stage, within the bottom panel.
- Using the Show References or Find Usages functionality.

**Finding Usages**

When you have selected a Data Stage within the Pipelines context, you can find out which other Data Stages use it by using the Find Usages feature available within the More button menu. Find Usages will list all Data Stages and all fields within those Data Stages that use the currently selected Data Stage. For example, if you utilized the Find Usages feature on a Data Store, you might find that it is used by a Data View - and within the Find Usages popup, you would see each specific field of the Data Store that is used by the Data View.

**Showing References**

The Show References feature is simply the opposite of the Find Usages feature described above. Continuing the example from the description of Find Usages, utilizing Show References on the Find Usages example’s Data View would should that the Data View references a Data Store; and, the Show References popup would list every Data Store field that the Data View references.

**Testing an Expression in the Expression Editor**

When building expressions throughout the product, you should make use of the collapsible Test panel shown on the right hand side of the Expression Editor. Once you have written an expression, this panel will allow you to experiment by entering values for fields used in the expression. Clicking Run Test will then output the result of the expression with those values in the bottom of the testing panel.
Additional Help Material

Listing of Screencasts

Here you can find a full listing of direct links to all Screencasts currently available to Infogix Data3Sixty Users and Administrators. Simply click a Screencast title to view it in your web browser. Screencasts are also accessible through use of Infogix Data3Sixty’s context sensitive Help button.

Administrator Screencasts

Administrator Setup
Shows Infogix Data3Sixty Admins how to get started by building Pipelines. (5:51)

Delegating Rights to Data Stages
Shows how to combine Environments, Users, Groups, and Super Groups to delegate access to Data Stages in Pipelines.

Custom Security
Shows how to assign Custom Security to Data Stages in Pipelines. (8:30)

Using the Executions History
Introduces the Executions History, which allows Infogix Data3Sixty Admins to track the executions of every Data Stage in a given Environment. (4:14)

User Screencasts

Introductory Content

Introduction to Infogix Data3Sixty
Provides a system overview of using Infogix Data3Sixty for Big Data processing. (5:38)

Acquiring your Data
Provides an overview of using Data Stores to acquire data. (3:23)
## Additional Help Material

### User Screencasts

#### Preparing your Data
Provides an overview of using Data Views and Analyses to prepare data. (4:02)

#### Analyzing your Data
Provides an overview of using Analyses, Dashboards, and the Infogix Data3Sixty Visualizer to analyze data. (10:51)

#### Acting on your Data
Provides an overview of using Infogix Data3Sixty’s collaboration features to act on data. (3:38)

#### Operationalizing your Data Stages
Provides an overview of using Process Models to operationalize data. (4:07)

#### About Pipelines
Provides an overview of using Data Stages within Pipelines. (6:07)

### Data Stores

#### External Data Stores
Explains how to create an external Data Store to access data already stored on Amazon S3. (5:33)

#### Upload Files to Data Store
Explains how to upload a file directly from your computer to create a Data Store. (5:05)

#### Linking Data Stores
Explains how to link multiple Data Stores so that they can be used together in a Data View. (7:47)

#### Enterprise Edition Data Stores
 Discusses the different Data Store types that are available when using the Enterprise Edition of Infogix Data3Sixty. (3:57)
### Analyses

**Join and Group By Analysis**
Covers the basics of using the Join and Group By Enhance nodes in an Analysis. (9:34)

**Data Enhancement Analysis**
Covers the basics of Data Store Inputs/Outputs and multiple Enhance nodes, including: Select, Rename, Sort, Filter, Union, and Distinct. (11:52)

**Intro to Co-Group and Flatten**
Introduces the some of the Analysis Designer’s more advanced Enhance nodes, Co-Group and Flatten. (4:16)

**Using the Bin Node**
Shows how to use the Bin Enhance node, to categorize numeric values within a given range. (4:07)

**Intro to the JavaScript Node**
Introduces the JS Enhance node, which enables custom scripting. (6:23)

**The In and Not In Nodes**
Compares the In and Not In Nodes, which combine aspects of the Join and Filter nodes. (7:59)

**Sampling in an Analysis**
Discusses the Sampling Node, and discusses how it is different from the Sampling Data for Testing settings found in Analyses. (5:15)

**The Auto Number Node**
Shows how to give a unique row number to each record in a dataset and persist these numbers in a field. (4:35)

**Reshaping a Dataset**
Demonstrates the Reshape Node, which can pivot columns to rows or rows to columns. (5:02)

**The Split Node and Line Parsing**
Introduces the Split Node, and how it uses Regular Expressions to parse fields. Also shows how to set up a Data Store as a single field that contains a line to parse. (6:17)
The SQL Node
Shows how to implement SQL within an Analysis, using the SQL node. (5:42)

The Execute Query in DB Node
Introduces the Execute Query in DB Node, which allows you to query an external database using parameterized values from other nodes. (5:01)

The Profile Data Node
Introduces the Profile Data Node, which allows you to generate information about a field in your data set. (5:47)

The Completeness Check Node
Introduces the Completeness Check Node, which allows you to check field values for predefined and custom conditions. (2:52)

The Consistency Check Node
Introduces the Consistency Check Node, which allows you to check values from multiple fields at once for custom conditions. (1:57)

The Timeliness Check Node
Introduces the Timeliness Check Node, which allows you to determine whether the time interval between two Date fields is acceptable. (1:41)

The Value Conformity Node
Introduces the Value Conformity Node, which allows you to determine whether values from a data set match specific values or fall within a range of values. (2:25)

Analytics

Overview of Analytic Models
Provides a broad overview of Analytic Models and the Analytics nodes, which enable machine learning. (4:35)

Overview of Segmentation
Introduces the basics of creating a Segmentation model to cluster a dataset. (8:04)

Overview of Classification
Walks through the basic work flow of Training, Evaluation, and Scoring a Classification model. (10:27)
Overview of Forecasting
Shows how to Forecast future values on time series data. Also introduces a method for splitting data into Training and Validation sets. (9:50)

Overview of Recommendation
Walks through the basics of creating a model that can recommend new products to users. (6:31)

Data Views

Creating a Data View
Explains how to create a basic data view, to enable visualization of data in Dashboards. (11:09)

Secure Fields
Discusses using Secure Fields in Data Stores, Data Views, and Dashboards. (15:18)

Dashboards

Create First Dashboard
Provides an introduction to creating your first Dashboard to visualize data. (12:49)

All About Filters
Discusses the different ways filters can be used when building and using Dashboards. (3:43)

Building Stacked Charts
Shows how to create a stacked bar chart, to visualize multiple series at once. (1:06)

Including Charts in Tables
Shows how to include charts as elements in your tables/data grids. (3:46)

Using Placeholders
Shows how to include Placeholders in Text dashlets, which can hold dynamic data values. (2:14)

Including Web Portals
Shows how to use the Web Content dashlet, to add a web portal to a Dashboard. (0:52)
Drilling to Another Dashboard
Shows how to make drillable charts link to other Dashboards when clicked. (3:55)

Using Geo Fields
Shows how to include Geo Fields in a Data View for use in Dashboards with maps. (7:10)

How to Make a Heat Map
Discusses the Heat Map charting type, which can be used to visualize multiple attributes at once. (2:11)

How to Make a Tree Map
Discusses the Tree Map charting type, which can be used to visualize multiple attributes at once. (3:15)

Time Series Charts
Discusses the Time Series charting type, which is useful when trying to represent chronological data. (3:39)

Window Functions
Introduces Window Function computations, by going over an example of Rolling Sum (5:03)

Waterfall Charts
Shows how to create a Waterfall Chart, to visualize cumulative values.

Charts for Stock Data
Shows how to build Candlestick and Open High Low Close charts, which are available within the Time Series charting option.

Dashboard Controls

Overview of Controls
Provides a basic overview of building controls to monitor your Dashboards. (4:49)

Western Electric and Nelson Rules
Discusses Control Charts’ predefined rule sets, Western Electric and Nelson. (5:55)
**Expression Thresholds**
Explains Expression Thresholds, which allow you to create simple expressions to monitor Dashboard measures. (4:01)

**Multipoint Thresholds**
Shows how to create Multipoint Thresholds, which allow you to apply a threshold to a series of points. (4:00)

**Visualizer**

**Customizing Chart Properties**
Shows everything that can be done with the Properties button, to customize the look and feel of your charts. (9:41)

**Query Criteria and Thresholds**
Discusses how to limit the amount of data queried from a Data View and set color coded Thresholds on charts when visualizing. (5:38)

**Showing Details**
Discusses how to Show Data Details when using any type of clickable chart. (2:41)

**Brush Select and Zooming**
Provides useful tips for chart navigation, and explains the difference between Brush Select and Zooming. (1:53)

**How to Create a Shared Axis**
Shows how to enable the Shared Axis option, when visualizing multiple computations at once. (1:08)

**Bubbles and other charting options**
Shows how to change charting display types, by clicking on a chart’s computations. (2:57)

**Using Gauges**
Discusses use of the Gauge charting type. (2:31)
Addtional Help Material

User Screencasts

Process Models

Basic Process Model
Introduces Process Model building, by automating the execution of a single Data Stage. (6:38)

Understanding Gateways
Discusses the Exclusive, Inclusive, and Parallel Gateways - all of which can be used to program the flow of Process Models. (9:31)

Connecting Process Models
Discusses use of the Execute Process Task to connect multiple Process Models. (3:20)

Property Reference Variables
Shows how to pass node variables across a Process Model, using Property Reference Variables. (2:33)

Notifying from a Data Store
Discusses using the Notify from Data Store Task, which can use a Data Store field to craft a message to be sent to users. (4:49)

Case Stores

Intro to Case Stores
Provides a brief introduction to the concept of a Case Store, which, once defined, can enable Case Management. (1:28)

Building a Case Store
Shows power users how to build out a case management UI for Case Work. (22:49)

Using a Case Store
Shows case workers how to use the case management UI. (4:44)