IBM Cúram Social Program Management

Cúram Intelligent Evidence Gathering(IEG)™

Version 6.0.4

Note

Before using this information and the product it supports, read the information in Notices at the back of this guide.

This edition applies to version 6.0.4 of IBM Cúram Social Program Management and all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Chapter 1

Introduction

1.1 Purpose

The purpose of this guide is to provide information on Cúram's Intelligent Evidence Gathering(IEG) component. This guide covers IEG script design, development, and execution.

Important

Please note this document covers a superseded version of IEG. This version of IEG is in maintenance mode and the new version of IEG is now the preferred technology for new development. Please refer to the Authoring Scripts using Intelligent Evidence Gathering(IEG) Developer's Guide for information on script design, development, and execution using the new technology.

1.2 Audience

This guide is intended for the business analysts and IEG developers who are responsible for the design and development of IEG scripts. It is assumed that the reader has a good understanding of the organization's evidence mapping processes and is familiar with the organization's evidence gathering processes. It is also assumed that the reader responsible for IEG development has Java, XML, web application, and CSS development experience.

1.3 Chapters in this Guide

The following list describes the chapters within this guide:

Intelligence Evidence Gathering (IEG) Overview

This chapter provides an overview of IEG including information on the

development process and on the IEG element structure.

Designing an IEG Script

This chapter provides the business analyst with an overview of the process of identifying and organizing the information that should be gathered during script execution. It also provides a set of guidelines for writing an IEG functional specification.

Creating an IEG Script

This chapter provides the IEG developer with a description of the process of creating an IEG script using the IEG Editor.

Looping in IEG

This chapter provides the IEG developer with a description of looping functionality.

Invoking an IEG Script

This chapter provides the IEG developer with information on how to invoke an IEG script.

Limiting IEG Script Modifications

This chapter provides the IEG developer with information on how to restrict users from making modifications to an IEG script during script execution.

The IEG Editing API

This chapter provides information on the use of the IEG Editing API to create and list remove IEG scripts and queston groups.

The IEG Execution Widget

This chapter provides information on the layout of the IEG Execution Widget. It also provides the IEG developer with information on how to configure IEG Execution Widget properties.

Using Gathered Evidence

This chapter provides the IEG developer with information on the IEG Execution API and on processing and storing information gathered during script execution.

Import and Export of IEG Definitions

This chapter provides the IEG developer with information on how question script and question group definitions may be imported to and exported from the database.

Chapter 2

Classic Intelligence Evidence Gathering(IEG) Overview

2.1 Introduction

Classic Intelligent Evidence Gathering(IEG) is an efficient alternative to traditional information gathering processes. With IEG, information is gathered interactively by displaying a script of questions that a user can provide answers to. Questions are only displayed if they are consistent with the user's previous answers so that the user is only required to provide answers relevant to his or her needs and situation. This creates a user-friendly environment that can be effectively implemented for a range of processes including client information intake, benefit assessment triage, online eligibility assessment, etc.

In contrast to traditional information gathering processes, IEG cuts down on the organization's administrative work by creating the potential for several routes through the same question script. This eliminates the necessity to develop many scripts for gathering information from different types of users.

A further advantage of IEG is the flexibility of its implementation and the range of its potential users. The IEG runtime environment can be set up for access from any UIM page. This means that IEG can be accessed directly from an organization application or remotely by an online user.

2.2 IEG Development

The two main components of IEG are the IEG Editor and the IEG Execution Widget. The IEG Editor allows the developer to define and maintain IEG elements including questions, question groups, preconditions, pages, and scripts. The IEG Execution Widget presents a dynamic set of pages to the user.

IEG can be easily developed by an organization for one or more of its in-

formation gathering processes. IEG development involves script design, creation, and implementation. The organization's business analysts design question scripts based on the information that must be gathered. An IEG developer can then define the script elements using the IEG Editor, configure the IEG Execution Widget, and invoke the script from the application. After invocation, the script can be accessed by users using the IEG Execution Widget.

The information collected at runtime can be extracted and processed or stored to a file or database. Because script execution is separated from the use of gathered information, the organization is provided with the flexibility of using gathered information when and how it chooses. Gathered information can also be stored indefinitely to satisfy traceability requirements.

2.3 The IEG Element Structure

The basic IEG elements are scripts, pages, preconditions, postconditions, question groups, questions, labels and sub-scripts. The IEG elements are logically organized into an element tree structure:

To summarize, IEG scripts consist of a hierarchy of elements structured something like this:

- Script
  - Page 1
    - Precondition
    - Question Group
      - Question 1
      - Label 1
      - Question 2
      - Question 3
      - Label 2
    - Page 2
    - Page 3
- Page 4
- Page 5
- Subscript

At the top of the tree structure is a script. Each script can contain one or more pages. Each page can contain a set of preconditions, a question group,

and child pages. Each question group can have one or more questions or labels, or any mixture of both.

Each of the following sections describes one IEG element.

2.3.1 Scripts

A script is an ordered set of pages. Although pages are ordered in a script, a page may or may not be displayed at runtime. For example, if Page 4 in the figure above is not displayed at runtime, the script order would be Page 1, Page 2, Page 3, Page 5.

Information that is external to a script may be passed to the script at runtime by using the RDO mechanism. When a script is defined, the RDOs that are accessible to the script are declared as attributes of the script. The qualified names of the data items of any declared RDO may be used anywhere question IDs may be used. They may form part of precondition expressions or default values etc. The RDO loader mechanism also provides IEG with the ability to invoke other Cúram functionality.

2.3.2 Pages and Child Pages

A page consists of a set of preconditions and a question group. The set of preconditions must be met in order for the page to be displayed. The question group consists of the questions that will be asked if the page is displayed. By combining preconditions and question groups, pages ensure that questions will only be asked if they are relevant to the user. For example, in the figure above, the user will not see the questions on Page 1 unless the preconditions for the page are met.

A page may have one or more dependent child pages. A child page has all the preconditions of the parent page and may also have additional preconditions. A child page will only be displayed if its parent page is displayed (and if any additional preconditions are met). At runtime, this ensures that child pages are not displayed unless the preconditions of both the child and parent pages are met. In the figure above, Pages 2 and 3 are child pages of Page 1.

2.3.3 Preconditions

A precondition is a set of boolean conditions that are based on the answers to previously asked questions. The evaluation of preconditions at runtime determines whether or not a page will be displayed during script execution.

2.3.4 Postconditions

A postcondition is a set of boolean conditions that are based on the answers to questions on the current page. The evaluation of postconditions at runtime determines whether or not the script execution moves beyond this page.

2.3.5 Question Groups

A question group is a logical set of questions that should be asked at the same time. Question groups can exist outside of scripts and can be reused across multiple scripts. At runtime, a question group appears as a list of questions and/or labels on a single page. Question groups may contain one or more questions or labels, or any combination of both.

2.3.6 Questions

A question is the most basic IEG element. It is used to collect a single piece of information. At script execution, each question is associated with question script text that phrases a question and a data entry field that allows the user to provide an answer in a specified format, e.g., a radio button or check box.

2.3.7 Labels

Labels are defined within question groups at the same level as questions. Labels are intended to provide information to users. A label can be hyperlinked. At runtime, a label appears as text (hyperlinked or otherwise) on a question page, not necessarily associated with any particular data-entry field.

2.3.8 Sub-scripts

Sub-scripts are simply IEG scripts included within a parent script or page. At runtime, a sub-script is executed as if its constituent parts were defined in the parent script.

Chapter 3

Designing an IEG Script

3.1 Introduction

The purpose of this chapter is to advise the business analyst on the IEG script design process. The information provided is intended as a guideline rather than as a set of instructions; the actual design process will be unique for each implementation of IEG.

For the design process to be successful, the IEG script design must create a bridge between the organization's business requirements and the implementation of a script. In practical terms, this means that the business analyst must create a functional specification that will be easily understood and implemented by the IEG developer.

3.2 Identifying the Purpose of the IEG Script

The first step in the IEG design process is to identify the purpose for running an IEG script. This generally involves an analysis of the results that are required from the script. The types of results that are required depends on whether the script is an intake or a triage script.

Intake IEG scripts usually have one result, i.e., the collection of relevant information. For example, if a client is found eligible for benefits, an intake IEG script can be used to collect additional information in order to determine the client's benefit amount.

Triage IEG scripts, in contrast, have more than one possible result. Triage IEG scripts are used to sort claimants into categories based on their needs and potential eligibility for benefits. After completing a triage IEG script, a claimant will have reached one possible result and will be placed into a specific category. For example, a triage IEG script may identify a claimant as potentially eligible for the non-resident commuter category of unemployment insurance. This would be one of several unemployment insurance eligible categories.

3.3 Identifying the Users of the IEG Script

An IEG script must be designed for use by a specific user type. It is important to thoroughly analyze the needs of this user type as this can impact the overall design of the IEG script. For example, an IEG script designed for use by case workers will be significantly different from an IEG script designed for use by a client. Each user type carries specific requirements for wording, layout, help functionality, etc.

It is also important to design an IEG script in relation to the way in which it will be accessed. A user accessing the IEG script from an organization location will have different requirements from a user accessing it from a personal computer at home. For example, a home user may require a more complete help structure than a case worker.

Depending on the type of user, localization might be required for a script. Script translations can be defined using the IEG Editor.

3.4 Identifying Previously Gathered Evidence

There are circumstances where evidence may have been previously gathered that may be relevant to an IEG script. Rather than gathering this evidence multiple times it may be made available to a script. This evidence may form part of a precondition or may be presented to the user for verification.

3.5 Identifying Required Cúram Functionality

There are circumstances where functionality provided elsewhere may need to be made available to an IEG script. For example, a user may be asked questions about their social security number, once this evidence is gathered it needs to be verified with an external agency and if it passes verification the IEG flow can continue. Access to external functionality is provided through RDOs and loaders.

Loaders are hand-crafted Java classes used to populate RDOs. If IEG tries to access a value of an RDO dataitem, the dataitem will first check if its value has already been loaded. If the value has not been loaded yet, the dataitem will invoke the loader. Answers supplied during script execution are available to loaders as they are registered in the Rules Context as they are encountered. See the `Cúram Rules Codification Guide` for more information on RDOs and loaders.

3.6 Phrasing Questions Based on Required Information

After identifying the purpose and users of the IEG script, it is important to

identify the information required by the organization. The process for identifying the information required involves an analysis of the conditions that must be met in order to reach the result(s) of the script. Each condition is phrased as a question. If designed well, the answers to these questions will provide the organization will all required information.

For example, a claimant who completes an unemployment insurance triage IEG script may be found potentially eligible or ineligible for unemployment benefits. Additionally, there may be several potentially eligible subtypes. One of these subtypes might indicate that a claimant is potentially eligible for unemployment insurance in a state as a non-resident commuter to that state. This would mean that the claimant lives in another state but regularly commutes to work in this state. This outcome is based on the person satisfying the following conditions during the claim period:

- The claimant is not resident in this state AND
- The claimant crosses a state line to commute to work AND
- The claimant has wages in only one state AND
- The claimant has wages in this state.

These conditions, phrased as questions, are included in the IEG script:

- Are you a resident of this state during the claim period?
- Do you cross a state line to commute to work during the claim period?
- Have you worked in more than one state during the claim period?
- Have you worked in this state during the claim period?

The process of analyzing the conditions that must be met to reach a result must be repeated for each possible result of the IEG script. When the conditions for reaching all possible results have been identified and phrased as questions, the designer can begin to organize these questions into an IEG script.

3.7 Grouping Questions

Related questions in a script should be grouped into question groups. For a user-friendly script, a question group should contain questions of a similar topic. It is important to note that the list of questions that must be answered for a particular result should not necessarily be grouped.

An analysis of all questions required for a script should be conducted so that questions can be logically grouped by topic. For example, the questions necessary for a unemployment insurance triage script should be listed and then sorted into topic-based groups such as resident state employment information, additional state(s) employment information, commuting information, etc.

Another important consideration when grouping questions is that questions that are dependent on each other should not be grouped. This is because it is impossible to limit the display of a question based on another question if both questions are in the same group. For example, a question determining whether a person commutes should not be grouped with questions about the commute. It is more effective to separate these questions so that a person who does not commute will not be asked questions that are irrelevant to his or her situation.

It is important to note that question groups should be short enough that they are easily manageable by the user. For example, an unemployment insurance triage may include many questions involving a person's state of employment. Asking all of these questions at once would result in a single page in the script containing an overwhelming number of questions. It is often effective to separate related questions into sub-groups that are displayed to the user in sequence as separate pages or child pages.

3.8 Ordering Questions, Labels and Question Groups

The order of questions and question groups in a script is important to a script's efficiency and user-friendliness. Both of these considerations must be taken into account when ordering questions.

To reach an outcome as quickly as possible, “prerequisite” information that immediately leads to a result should be collected early in an IEG script. Prerequisite information includes information that determines initial potential eligibility or ineligibility. For example, a person is ineligible for unemployment insurance if he or she has been determined ineligible for another unemployment insurance claim in the last quarter. If this information is collected early in the script, it may be possible to complete the script without asking unnecessary questions.

Reaching an outcome as quickly as possible must, however, be balanced by the necessity to develop a user-friendly script. If all prerequisite information is collected at the beginning of the script, the script might become off-putting for the user or lead to confusing results when the script is ended. For example, a person may be ineligible for unemployment insurance if he or she has a disability, but it is not necessarily appropriate to ask a person about his or her disability at the beginning of a script.

The organization may also wish to make the ordering of a script more or less transparent based on the user type. For example, if a caseworker is running a script that is ended after a question group is presented, it should be clear which question caused it to end. That is, two prerequisite questions should not be asked in a row as this could lead to confusion as to which question ended the script. On the other hand, if a claimant is running the script from home, it may not be suitable for him or her to see exactly how an outcome was reached.

Within the IEG Editor, when viewing a question group, the order in which the questions/labels are displayed corresponds directly to the order they are

asked within the IEG Script. To change this order, simply select the desired action, Up or Down, associated with the question/label you wish to move. Repeat this process until the desired order is achieved.

3.9 Defining Preconditions

The circumstances under which pages will be displayed is another important design consideration. The business analyst can define these circumstances by defining preconditions for pages within a script (see Section 3.11.2, *Designing Pages and Question Groups*). Because preconditions limit the display of question groups rather than of individual questions, entire groups of questions can be eliminated from a script execution based on the answer(s) to a previous question(s). This allows the same script to be easily used for a wide range of users. For example, if a claimant completing an unemployment insurance triage script answers that he or she has only worked in one state, a group of questions on multiple employment states will not be displayed to that claimant. This group will, however, be displayed to users who have worked in more than one state.

Properly defined preconditions will ensure that questions are not posed unnecessarily. For each question in a script, the business analyst needs to map the relationship between that question and every question group that occurs later in the script. If the question has a significant impact on the relevancy of a question group, a precondition should be defined. This ensures that the question group will not be displayed if it is irrelevant to a user.

When defining preconditions, it is important to ensure that each page in a script will be displayed under an achievable set of circumstances. That is, preconditions should not be so limiting that it is impossible for a user to reach a page.

3.10 Defining Postconditions

Postconditions are used to perform page validation. Script execution cannot proceed unless all postconditions defined for a page are satisfied. Each postcondition that fails will have an error message displayed. Postconditions are particularly useful to check that the answers to the current page are compatible with answers from previous pages.

3.11 Writing an IEG Functional Specification

After the design process has been completed, a functional specification should be written. The functional specification should include all information required by an IEG developer to define the IEG script using the IEG Editor. This information should include an identification of all elements contained within a script and a brief description of its purpose and intended use. It should also define the layout of each page in the IEG runtime environment and any information passed into the script.

Generally, the functional specification will contain an introductory chapter that identifies the purpose of the script, the users of the script, the possible outcomes of the script, the information made available to the script and the basic page layout of the IEG Execution Widget. It should also outline the requirements for the finish page that will be displayed after a script has been completed.

After the introductory chapter, a chapter is usually created for each page in the script. This creates a table of contents that clearly represents the progression of the script from the start to all its possible results. This layout also helps the IEG developer to develop the script page-by-page, which is the most effective way of using the IEG Editor.

3.11.1 Specifying the Basic Page Layout for IEG Execution Widget

The basic page layout for the IEG Execution Widget should be outlined in the functional specification. Four panels can be configured for pages in a script. These are the tab panel, question script panel, question panel, and navigation panel. The specifications for each panel should be provided. (See Chapter 9, *The IEG Execution Widget* for more information on the IEG Execution Widget.)

3.11.2 Designing Pages and Question Groups

The requirements for each page in a script must be documented. Each page can have several attributes associated with it. These attributes help the IEG developer develop the script using the IEG Editor and define the way in which the script will execute at runtime.

Note that a question group can be reused across scripts. This preserves the question group name, description, and the questions contained within it. It does not, however, preserve the preconditions, loopsize, etc., because these are associated with pages rather than question groups. Note also that a page does not have its own name or description. The name and description of the associated question group are used instead.

The following list includes each attribute that can be maintained for a page. If one of these attributes is not needed for a particular IEG script, it is generally not documented in the functional specification.

Page Name

The page name is used to identify the page in the IEG Editor. If no page name is specified, the page assumes the name of its associated question group both in the IEG Editor and at runtime.

Question Group Name

The question group name is used to identify the question group in the IEG Editor. Note that the question group name becomes the page name at runtime if no page name is specified. For example, a question group

may be named “Employment State Information.”

Question Group Description

The question group description is used to describe the question group. Note that the question group description becomes the page description at runtime. The description can be used as needed by the organization. For example, the description might be used as an overview of the questions that will be asked or it could be read aloud by a case worker as an introduction to the questions that will be asked.

Child Page

A child page shares the preconditions of its parent page. It may also have additional preconditions. It is important to document whether or not a page is a child page as this has implications for the way in which the IEG developer will create the page in the IEG Editor.

Loopsize

A loopsize can be defined for a page. The loopsize refers to the number of times a question group should be displayed to the user. The loopsize is based on a numerical answer given for a previously asked question. For example, if a person specifies that he or she has worked in four different states, a page that collects information on states of employment may be displayed four times. Where a page has child pages, the question group on the page will be displayed the loopsize number of times. Then, the script will iterate through the entire sequence of child page question groups the loopsize number of times.

Legislation Link

If relevant legislation exists, the legislation link allows the user to access the URL that contains this legislation.

Policy Link

If relevant policy exists, the policy link allows the user to access the URL that contains this policy.

Preconditions

The preconditions for displaying a page should be defined. The preconditions relate to specific questions contained on earlier pages. For example, a precondition relating to the question, “Do you work in this state?” might read, “The client works in this state.” It is also helpful to include the relevant question number and page in the precondition so that the question can be easily referenced.

Postconditions

The postconditions for leaving a page should be defined. The postconditions relate to specific questions contained on earlier pages and the current page.

3.11.3 Defining Questions

The requirements for each question must be documented. Each question can have several attributes associated with it. These attributes help the IEG de-

veloper define the script using the IEG Editor and define the way in which the script will execute at runtime.

Questions are defined within the chapter for the page that they will appear on. Each question is also defined as part of a question group. Note that questions can not be maintained independently of a question group and cannot be reused across scripts.

Textual descriptions of questions (labels, descriptions, texts) can include variables, e.g. text based on the answers to questions on previous pages. For example, if a question of a question group on Page 1 asks the user's name, the answer given can be used in the description of a question included in a Question Group on Page 2.

To use this feature, the developer simply includes a reference to the required variable in the value attribute of the desired textual description. If ther user answers the question "What is your name?" with "John smith", then on a following question page a question could resolve to "Address for John Smith" by referencing a textual value of the original question:

```
Address for <PersonDetailsGroup.PersonName>
```

Questions can be assigned an identifier or number by the business analyst. This is helpful in documenting preconditions and loopsizes. Note that this number may differ from the question ID that will be assigned by the developer in the IEG Editor.

The following list includes each attribute that may be maintained for a question. If one of these attributes is not needed for a particular IEG script, it is generally not documented in the functional specification.

Question Name

The question name is used to identify the question in the IEG Editor. For example, “Employment State.”

Question Script

The question script is the actual question text that will be displayed at runtime. For example, “Do you work in this state?”

Help Text Script

Help text script provides the user with support for answering the question. For example, “This is the state where you earn wages.”

Question Alias

A question alias is an alternative phrasing of the question script that can be displayed at runtime.

Answer Data Type

The answer data type defines the data type that the answer will be given in, e.g., a boolean, money, or string value. (A full list of answer data types is provided in Appendix B, *Answer Data Types*.) Note that additional answer data types can be added if required by the organization.

The answer data type selected for a question is usually the data type that will be required for processing the answer after script execution (see Section 3.13, *Specifying the Use of Gathered Information*).

Data Entry Field Type

The data entry field type defines the kind of entry field that will be provided for a question. The data entry field may be a check box, radio buttons, a text entry field, or a date entry field (with a pop-up calender widget). Note that more than one data entry field type may share an answer data type. For example, check boxes and radio buttons can both return boolean values.

Default Answer

The default answer defines the answer value that will be used if no answer value is entered by the user, e.g., False, True.

Mandatory Indicator

If the mandatory indicator is selected for a question, an answer will have to be provided for the script to continue past the question. Note that depending on the answer data type and the data entry field, the user may not have to perform any action to answer a question. For example if the default answer is False (unselected) and the data entry field is a check box, the answer will be read as False unless the user checks the box.

Record Unanswered Indicator

If the record unanswered indicator is selected, a check box will be provided next to the question at runtime. If a client refuses to answer the question, then the user must check the box in order for the script to continue past the page. Note that the record unanswered indicator is generally only used when the user is a case worker.

Legislation Link

The legislation link allows the user to access the URL that contains the legislation relevant to the question.

Policy Link

The policy link allows the user to access the URL that contains the policy relevant to the question.

3.11.4 Defining Labels

Each label is defined as part of a question group. Note that labels can not be maintained independently of a question group and cannot be reused across scripts.

Each label has attributes specifying whether or not it is a hyperlink and the URL to use if it is a hyperlink. Textual aspects of labels (label texts, URLs) may include substitution expressions. The substitution expressions can reference RDO data items and questions previously encountered in the script. For example, if a question in a question group asks the user's name, the answer to this question can be used in the URL of a label included in sub-

sequent question group.

To use this feature the developer simply includes a reference to the required question, enclosed in angle brackets, as part of the URL. For example, if the user answers the question "What is your place of birth?" with "Dublin", then on a following question page a label URL could resolve to "http://.../page2.do?birthpl=Dublin" by referencing a textual value of the original question.

```
http://.../page2.do?birthpl=
                    <PersonDetailsGroup.BirthPlace>
```

The following list includes each attribute that may be maintained for a label. If one of these attributes is not needed for a particular IEG script, it is generally not documented in the functional specification.

Text

The text is the actual label text that will be displayed at runtime. For example, “All questions are mandatory.”

URL

The URL attribute is the URL that the label will link to in the event that the Hyperlink flag is enabled. The text is displayed, and the URL is attached to it For example, “http://.../IEGScripts/question_information”

Hyperlink

This is a boolean flag that specifies if the Text of a Label will appear as a hyperlink or as plain text. By default, it is set to true, enabling the hyperlink.

3.11.5 Designing the Finish Page

The requirements for the UIM finish page should be defined in the functional specification. The organization can add whatever functionality it requires to the finish page.

3.12 Defining a Reference to a Sub-Script.

From within a Script it is possible to create a reference to another Script, referred to as a Sub-Script. Outside of the context of the enclosing Script a Sub-Script is a stand alone Script that can be executed independtly. Consequently if the Sub-Script is modified it will impact any Script that includes a reference to it.

It is possible for an enclosing Script to refer to the Questions and Labels of a Sub-Script. However since a Sub-Script remains independent from the enclosing Script, it is not possible for a Sub-Script to reference the enclosing Script.

A reference to a Sub-Script can be created at the Root or Page level of a

Script. It should be noted that a Sub-Script can only be referenced once within a Script. A Sub-Script should not include a Question Group that has already been included elsewhere within the encluding Script.

3.13 Specifying the Use of Gathered Information

The information gathered from the execution of an IEG script can be used as required by the organization (see Chapter 10, *Using Gathered Evidence*). Cúram does not specify how information should be used but provides the facility to store information indefinitely or to access information for a specific purpose.

Within the Cúram framework, information gathered using IEG can be mapped onto Cúram case evidence (effective from a specific date). If a case does not exist at execution time, information may be stored or sent for review by a caseworker.

If the use of gathered information is specified during the design stage, the script can be strengthened to better fulfill the requirements of that use. For example, if data collected during a script execution is to be used as case evidence, answers can be formatted to feed directly into a case.

Chapter 4

Creating an IEG Script

4.1 Introduction

This chapter provides information on using the IEG Editor to create and edit IEG scripts. Using the functional specification written by the business analyst, the IEG developer will have the information required to maintain IEG scripts, pages, preconditions, question groups, questions, labels, translations, and expressions.

4.2 The IEG Editor Script Tree View

The IEG Editor provides a tree view that allows the IEG developer to easily navigate among the elements contained within a script.

The view page for each element can be accessed by clicking on its name. The currently selected item is indicated on the tree with an underline. In the figure above, Question 1 is the currently selected element.

Note that question translations, label translations, question group translations, and script translations do not appear in the tree view. These can be maintained from the relevant question, question group, or script.

4.3 Maintaining Scripts

When a script is created, it is assigned a script ID by the IEG developer. The ID must be a unique, alphanumeric value with no spaces. Each script must also be assigned a script name and description. These are specified in the functional specification.

In order to allow for multiple versions of a basic script, using the IEG Editor it is possible to create a script by cloning another. However it is important to note that this process does not perform a 'deep' clone. Effectively the script is cloned; however the constituent question groups are not. This is because

scripts have 'pointers' to question groups, not actual copies of each individual question group.

Scripts may also be created using the IEG Editing API where a type for the script may be optionally specified. Scripts defined with a type are not displayed when listing scripts in the administration application. However a list of scripts of a given type can be obtained using the IEG Editing API, see (see Section 8.2, *Creating a Script* for more information.)

For a script to be successfully executed at runtime, it must contain at least one page. Each page must additionally have one preconditions section (containing zero or more conjunct preconditions), one question group, and zero or more child pages. Each question group must also have at least one question.

4.4 Maintaining Pages

Pages can be created for a script.

A loopsize can be defined for a page. The loopsize attribute is specified as an expression that relates to the answer provided for another question. For example, the loopsize attribute may appear something like “QGRP1.Q3”, or “QGRP1.Q3 - 1”. The loopsize attribute specifies the control variable for a loop. For more information on creating expressions using the IEG Editor, see Section 4.11, *Defining Expressions and Using the Formula Helper*.

Notes can also be defined for a page. The note text is specified in the functional specification.

Legislation links and policy links can be defined for a page. These are specified in the functional specification and are absolute URLs linking to the relevant legislation/policy.

Note that at runtime, pages inherit their names and descriptions from their associated question group.

4.4.1 Maintaining Child Pages

One or more child pages can be created from another page or child page. The process for creating a child page is identical to creating any other page; loopsize, preconditions, notes, and legislation/policy links can all be defined for a child page.

The advantage of creating child pages is that the developer will not have to recreate the preconditions or loopsize of the parent page (those of the parent page are automatically shared with its child pages). Note that additional preconditions or another loopsize can be added to a child page, if necessary.

4.5 Maintaining Preconditions

A precondition can be created for a page. Preconditions are defined as ex-

pressions. A sample precondition may read “QGRP1.Q3 == true ”. For more information on creating expressions using the IEG Editor, see Section 4.11, *Defining Expressions and Using the Formula Helper*.

4.6 Maintaining Postconditions

Multiple postconditions can be created for a page. Postconditions are defined as expressions. A sample postcondition may read “QGRP1.Q4 <= 15 ”. For more information on creating expressions using the IEG Editor, see Section 4.11, *Defining Expressions and Using the Formula Helper*.

Each postcondition is associated with an ID (a meaningful description) and a message (and its various translations) to be displayed if the postcondition is false.

4.7 Maintaining Question Groups

There are three ways of creating a question group. The first uses the IEG Editor to create a question group for a page. The second creates a question group in the administration application. The third is through the IEG Editing API. In all cases question groups are stored on the system so that they can be reused across scripts.

Each question group must be assigned a question group ID that is created by the IEG developer. The ID must be a unique, alphanumeric value with no spaces. Each question group must also be assigned a name and description. These are specified in the functional specification. Note that the question group name and description are displayed as the page name and description at runtime.

Question groups can be added to pages within the IEG Editor. A question group created in the IEG Editor is added to the page it was created from. An existing question group can also be added to a page. A selection page allows the developer to select an existing question group from all stored question groups. Note that the same question group cannot be added to more than one page in a single script and that the selection page will not list question groups that are already in the script.

Using the IEG Editor it is possible to reorder questions simply by moving each individual question up or down within a question group. (see Section 3.8, *Ordering Questions, Labels and Question Groups* for more information.)

Question groups created with the IEG Editing API may have a type for the question group optionally specified. Question groups defined with a type are not displayed when listing question groups in the administration application. However a list of question groups of a given type can be obtained using the IEG Editing API, see (see Section 8.5, *Creating a Question Group* for more information.)

Question groups can be maintained from both the IEG Editor and the ad-

ministration application. If a question group is modified in one location, all other instances of the question group will also be modified.

Question groups that are used in IEG scripts cannot be deleted. To do that you need to detach question group from script. There are a few ways to do that:

- Delete question page;
- Update the question page in the IEG script to reference another question group;
- Delete question script (not recommended unless absolutely necessary).

4.8 Maintaining Questions

Questions can be added to question groups using the IEG Editor. Each question must have a minimum of an ID and an answer data type defined. The question ID is created by the IEG developer and must be a unique, alphanumeric value with no spaces. The answer data type is used by the client infrastructure to determine the type of control that should be used to record the answer. An answer data type can be selected from a list of answer data types provided in the IEG Editor. If required, additional answer data types can be added to this list (see Appendix B, *Answer Data Types* for more information). If the answer data type is a string, the input field presented to the user can consist of multiple lines (the default is one).

If the selected answer data type is a CODETABLE_CODE, then the runtime control displayed will be a drop-down list of possible answers. List meta data can be specified to indicate the inclusion of a blank default row and the number of visible rows. Other meta data settings indicate whether the list should be single-select or multi-select, i.e. in the case of multi-select, the user has the option to choose more than one entry from the list by holding down the Ctrl key and clicking on the required answers.

A default value can also be defined for each question. If defined, the data entry fields for this question will be prepopulated with this value upon script execution. The default value is defined as an expression. A sample default value may read “False”. For more information on creating expressions using the IEG Editor, see Section 4.11, *Defining Expressions and Using the Formula Helper*.

The mandatory and record unanswered indicators should be selected as indicated in the functional specification. Script text and question help text can also be added if specified in the functional specification.

Legislation links and policy links can be defined for a question. These are specified in the functional specification and are absolute URLs linking to the relevant legislation/policy.

Question aliases can be defined for a question. Question aliases are an alternative phrasing of the question, depending on the target audience of the

script. When an alias is created for a question, the type of alias is selected from a list of types maintained in the AliasType code table.

Note that if a question is referenced in a precondition, loopsize, default answer expression or postcondition, it cannot be deleted from a script until the reference is deleted.

4.9 Maintaining Labels

Labels can be added to question groups using the IEG Editor. At a minimum, each label must have an ID and a Text defined. The label ID is created by the IEG developer and must be a unique, alphanumeric value. It can contain spaces. Once created, it cannot be modified.

A URL can be defined for a label. It is possible to use variable substitution in this URL, including answers to questions defined previously in the IEG script. When checked, the Hyperlink checkbox renders the Text as a Hyperlink to the specified URL at runtime. When unchecked, Text appears as plain text.

4.10 Translating Script Elements Into Other Languages

Script elements are initially created using the default server language. Script elements can subsequently be localized so that the script can be executed in multiple languages without the need to redevelop it.

A list of translations is maintained for each script element. Within an element, a translation can be maintained for any field that contains text, e.g., question text, help text, etc. To create a translation from a script element, the language of translation must be selected from a list of languages. These languages are maintained in a code table as part of application administration. For each script element, only one translation is allowed per language.

At runtime, the script is localized according to the user's locale. For more information on localization, see the Cúram Administration Guide.

4.11 Defining Expressions and Using the Formula Helper

Expressions are used to define precondition, loopsize, and question default answer values. Expressions consist of one or more clauses which evaluate to a discrete value (in the case of preconditions, a boolean value; in the case of loopsize, an integer; in the case of question default values, a value appropriate to the Question Type). Boolean expressions use case sensitive “and”, “or” and “xor” conjunctions, while other expressions use comparison operators (==, <=, >=, >, <, !=) and arithmetic operators (+, -, /, *). Expressions can contain an arbitrary number of clauses and can use opening and closing brackets to control precedence. Expressions can also contain constant values

and the answers to previously asked questions. For more information on the operations that are supported for IEG expressions, see Appendix A, *Operations Supported for IEG Expressions*.

A single text field is provided for entering a precondition, loopsize, or question default value expression. This allows the IEG developer to enter the expression directly into the field. A formula helper is provided to assist in expression creation. The formula helper allows the IEG developer to search for either a question group or an RDO. If a question group is searched for the formula helper returns a list of all questions in that group. The IEG developer can copy the question ID of any of these questions to the clipboard and then paste the ID into the expression text field. If an RDO is searched for the formula helper returns a list of all data items in that RDO. The IEG developer can copy the data item name of any of these data items to the clipboard and then paste the ID into the expression text field.

A question ID used in an expression must be for a previously asked question. An error will be thrown if an IEG developer attempts to insert a question ID that does not occur earlier in the script. There are two circumstances, however, when an expression cannot be validated until runtime. First, the question default value formula helper is called for a question group which may have been created outside of a script or may have been included in many scripts. As such, it is not possible to validate which questions have been asked previously until runtime. Second, an expression may include question IDs for questions that are not displayed during a script execution because the preconditions for the page on which they are contained were not met. If this happens, the expression will be invalid at runtime.

An expression added within a string value for default answer will not be evaluated. The expression will instead be dealt with as part of the string.

Chapter 5

Looping in IEG

5.1 Introduction

This chapter provides information on IEG's looping functionality. It sets out to describe the various kinds of loops that can occur in execution; and to explain what situations will produce each particualar loop-type and why.

The loop types supported by IEG are:

- FOR loop;
- WHILE loop;
- FOR-EACH loop;

5.2 Loopsize expressions

The loopsize attribute is specified as an expression that relates to an answer provided for another question, a dataitem of an RDO or a literal. For example, the loopsize attribute may appear something like “QGRP1.Q3”, or “QGRP1.Q3 - 1”. Loopsize expressions determine the number of times question pages and questions are presented to the user. For more information on creating expressions using the IEG Editor, see Section 4.11, *Defining Expressions and Using the Formula Helper*.

5.3 FOR loop

IEG's FOR loops are based on a loopsize expression that evaluates to an integral value. Depending on the type of For loop either the question page itself or the questions on the page will be repeatedly displayed a set number of times. Irrespective of the FOR loop type the child pages of the question page will also be repeatedly displayed.

The number of iterations of a FOR loop can only be changed by changing the control variable, i.e. supplying a different answer to the loop question. The loopsize expression is resolved before the first iteration of the loop

There are two types of FOR loop available in IEG. The type of FOR is controlled by the following boolean property: `curam.iegruntime.questionpage.separatequestionsforloopstyle`

5.3.1 Exclusive FOR loop

IEG's Exclusive FOR loop means that, if a page contains a integral loopsize expression, the questions on that page will appear multiple times. The page itself only appears once. The Exclusive FOR loop is the default FOR loop type in IEG.

5.3.2 Inclusive FOR loop

The Inclusive FOR loop means that the question page that contains the loopsize expression forms part of the loop. The page itself is displayed multiple times and the questions on the page are not repeated. The Inclusive FOR loop will be adopted as the looping style if the boolean property `curam.iegruntime.questionpage.separatequestionsforloopstyle` is set to true.

5.4 WHILE loop

The WHILE loop bases its loop condition on any boolean expression, often the answer to question asked within the loop. If care is not taken when defining a WHILE loop an infinite loop may result.

In this case the loopsize expression is referencing a question within the loop, with the answer indicating whether the loop should be iterated over again, e.g. when compiling details for a household member, a question of boolean answer data type asks "Are there any more people in the household?". The size of such a WHILE loop can be changed by revisiting the loop in execution and answering the control question differently.

5.5 FOR-EACH loop

A FOR-EACH loop bases its loop condition on iterating across all items of a list. In this case, a loopsize expression resolves to the answer(s) given to a question that allows multiple answers, i.e. answers selected from a codetable.

The size of FOR-EACH loops can only be changed by changing the size of the control variable, i.e. the list.

Nested FOR-EACH loops present their own issues. When declaring a ques-

tion page in a loopsize, the for-each loop will iterate the number of times that the original page was looped. If this page contains sub-pages, in order to access particular instances of the sub-page a nested loop must be set up within the parent loop. If one tries to access an instance of nested sub-page while not operating within a loop of the parent, access will only be given to the last iteration of the nested sub-page.

5.6 Nested loops

A nested loop is a loop within a loop; an inner loop within the body of an outer one. In IEG, the first pass of the outer loop triggers the inner loop, which then executes the list to completion. The second pass of the outer loop triggers the inner loop again, and this repeats until the outer loop finishes.

Questions are registered one instance at a time - therefore, in a nested loop, there is no way to refer to questions in different iterations of the enclosing loop.

Chapter 6

Invoking an IEG Script

6.1 Introduction

When a valid script has been created using the IEG Editor, the script may be executed. Script execution is controlled by the `IEGRuntime` class. The `IEGRuntime` class also provides facilities to list the scripts that are available for execution, obtain the result of a script execution, and remove a script execution when the script has completed.

6.2 Listing Available IEG Scripts

The `listScripts` method of the `IEGRuntime` class will return details of the IEG scripts that are currently available for execution. The method returns a ScriptList struct which is a list of scriptDetails structs. The *`scriptDetails`* struct has two members. The ID of a script is held in the *`scriptID`* member and the name of a script is held in the *`scriptName`* member.

The following example shows how to list the details of scripts that are available for execution:

```
// List available scripts.

final IEGRuntime runtime = IEGRuntimeFactory.newInstance();
ScriptList scripts = runtime.listScripts();
int numberOfScripts = scripts.dtls.size();
ScriptDetails scriptDetails = null;
for (int x = 0; x < numberOfScripts; x++) {
    ...
    scriptDetails = (ScriptDetails) scripts.dtls.get(x);
    ... = scriptDetails.scriptID;
    ... = scriptDetails.scriptName;
}
```

6.3 Initiating a Script Execution

To execute a script, the script must be identified and its execution ID obtained. To initiate the script execution, the script ID is passed to the `createScript` method of the `IEGRuntime` class. This method returns a generated unique execution ID. The execution ID is then subsequently used to refer to that particular instance of the script execution.

Two struct types are used with the `createScript` method, ScriptIdentifier and ExecutionIdentifier. The script ID is assigned to the *`scriptID`* member of a ScriptIdentifier struct to be passed to the method. The execution ID is contained in the *`executionIdentifier`* member of an ExecutionIdentifier struct that is returned by the method.

The following sample server method will initiate a script execution. The method is passed the script ID of the script that is to be executed.

```
// Initiate script execution.

public class ServerClass implements ...
{
 public ExecutionIdentifier setupMethod (...
  ...
  final ScriptIdentifier scriptIdentifier =
  new ScriptIdentifier();
  scriptIdentifier.scriptID = ... // id of the script to execute.
  final IEGRuntime iegRuntime = IEGRuntimeFactory.newInstance();
  ExecutionIdentifier executionIdentifier =
  iegRuntime.createScript(scriptIdentifier);
```

6.3.1 Initializing a Script Execution From a UIM Page

Script execution is initialized from a UIM page. While the IEG Execution Widget controls the script execution, it does not initiate it. Therefore, the script execution is initialized on one UIM page and the execution ID is then passed to another UIM page that contains the Execution Widget.

The following is an example of a UIM page that contains a script execution initiation button. The ID of the script to be executed is passed to the UIM page as a page parameter. The script ID is passed to the initialization code and the execution ID is obtained. The execution ID can then be forwarded to the page that contains the IEG Execution Widget.

```
<!-- Initial Script Execution UIM page -->

<SERVER_INTERFACE OPERATION="setupMethod"
           CLASS="ServerClass"
           NAME="ExecutionSetup"
           PHASE="ACTION" />

<PAGE_PARAMETER NAME="scriptID" />

<CONNECT>
  <SOURCE PROPERTY="scriptID" NAME="PAGE" />
  <TARGET PROPERTY="scriptID" NAME="ExecutionSetup" />
</CONNECT>

<ACTION_SET>
  <ACTION_CONTROL LABEL="ActionControl.Label.Execute"
           IMAGE="Button.Execute"
           TYPE="SUBMIT">
    <LINK PAGE_ID="IEG_PlayerPage">
```

```
      <CONNECT>
        <SOURCE NAME="ExecutionSetup" PROPERTY="executionID" />
        <TARGET NAME="PAGE" PROPERTY="executionID" />
      </CONNECT>
    </LINK>
  </ACTION_CONTROL>
</ACTION_SET>
```

The main parts to the above example UIM page are as follows:

- A SERVER_INTERFACE is defined to call the execution initialization code (in this example the setupMethod method of the Server-Class class) and return the results in a struct called ExecutionSetup.
- A PAGE_PARAMETER is passed to the page called “script”.
- Once the page is created, the page parameter containing the script ID is passed to the setupMethod via the first CONNECT.
- An ACTION_CONTROL which is part of an ACTION_SET is defined to provide a button used to confirm that the script is to be executed.
- When the button is pressed, the server interface method is invoked and the execution ID is obtained and passed to the page that contains the Execution Widget via the second CONNECT: In this example, the execution ID is passed to the page IEG_PlayerPage in the page parameter called *executionID*.

6.3.2 Passing the Execution ID to the IEG Player Widget

The IEG Execution Widget must be supplied the execution ID of the script execution that it will assume control of.

The following is an example of a UIM page that contains the IEG Execution Widget. The execution ID is passed to the UIM as a page parameter. The WIDGET tags indicate a Widget of type IEG_PLAYER is being used. The Widget accepts a single parameter, *EXECUTION_ID*, supplied from the page parameter *executionID* via the CONNECT.

```
<!-- Passing execution id to IEG Player Widget -->

<PAGE_PARAMETER NAME="executionID"/>

<WIDGET TYPE="IEG_PLAYER">
  <WIDGET_PARAMETER NAME="EXECUTION_ID">
    <CONNECT>
      <SOURCE NAME="PAGE" PROPERTY="executionID"/>
    </CONNECT>
  </WIDGET_PARAMETER>
</WIDGET>
```

Once the IEG Player has been invoked, it will assume control of the script execution and ensure that all script criteria defined in the IEG Editor are implemented. Navigational functionality allowing the user to navigate through pages is inherent in the Widget so that no additional conditional code is required.

6.4 Continuing an Interrupted Script Execution

If a script execution has been interrupted due to a session timeout, browser crash, etc., before it has completed, the questions already answered will still be available on the database. In order to continue with the execution of an interrupted script, the execution ID is required. As in Section 6.3.2, *Passing the Execution ID to the IEG Player Widget*, the execution ID of the selected script is passed to the UIM page that contains the Execution Widget (IEG_PlayerPage in the previous example). Script execution will resume at the last page that was presented to the user.

Since IEG is intended to be used in a number of different circumstances, no assumptions can be made that script executions are associated with a particular case. It is the responsibility of the application developer to implement a mechanism to maintain a link between the execution ID and information to help the user identify a particular script. That association should allow the user to identify an execution ID, to then pass into the appropriate UIM file and restart the desired script.

6.5 Script Execution Status

The `IEGRuntime` class contains a method `isExecutionCompleted` that will indicate if the script execution has finished. Execution is finished if there are no further pages to be presented to the user.

The execution ID of the selected script is passed as a parameter to the `isExecutionCompleted` method.

```
// Script execution status.

final ScriptIdentifier scriptIdentifier =
  new ScriptIdentifier();
scriptIdentifier.scriptID = ... // id of the script to execute.
final IEGRuntime iegRuntime = IEGRuntimeFactory.newInstance();
ExecutionIdentifier executionIdentifier =
  iegRuntime.createScript(scriptIdentifier);
...
boolean finished =
  iegRuntime.isExecutionCompleted(executionIdentifier)
```

6.6 Script Execution and RDO Support

As a script is executing the RDOs that are declared to be accessible to the script are loaded as their data items are encountered. For example, if a precondition refers to a data item of an RDO, that RDO and all of its data items are loaded into memory. As subsequent data items are encountered their values have already been loaded so they are just retrieved from memory. If a user pages backwards in the script, new values for the data items will not be loaded unless the user pages back to a point before the RDO was first referenced. Therefore if the user pages backwards to a point where the RDO is

reloaded, new and up to date values can be obtained for the data items.

6.6.1 Passing Preinitialized RDOs to a Script

RDOs may be created, initialized and passed to a script when the script is being created. The RDOs are passed as a list, along with the script ID to the `createScript` method of the `IEGRuntime` class. The `createScript` method that accepts a list of RDOs as a parameter is available when an `IEGRuntime` object is instantiated directly rather than using the `newInstance` method of the `IEGRuntimeFactory` class. The `createScript` method returns a generated unique execution ID. The execution ID is then subsequently used to refer to that particular instance of the script execution. Loaders are not invoked for RDOs that are preinitialized.

```
// Initiate script execution.
public class ServerClass implements ...
{
 public ExecutionIdentifier setupMethod (...
  ...
  // Create the preinitialized RDOs to be passed to the script.
  final ArrayList rdoList = new ArrayList();
  final ItemGroupGlobals globals = new ItemGroupGlobals();
  ... // initialize the RDO
  rdoList.add(globals);
  ...

  // Create the ID that identifies the script to be executed.
  final ScriptIdentifier scriptIdentifier = new ScriptIdentifier();
  scriptIdentifier.scriptID = ... // id of the script to execute.

  // Create the IEG runtime object used to create
  // the script execution.
  final IEGRuntime iegRuntime = new IEGRuntime();

  // Create the ID that identifies the script execution.
  final PlayerExecutionByID playerExecutionByID =
    new PlayerExecutionByID();

  // Invoke the create script method passing the list
  // of preinitialized RDOs.
  playerExecutionByID.playerExecutionID =
    iegRuntime.createScript(scriptIdentifier, rdoList).executionID;
```

6.6.2 Accessing Answers in Loaders

While the majority of loaders are paired with BPOs to retrieve information from the database, loaders are simply Java classes and can implement any functionality required and retrieve information from a range of other sources. As script execution proceeds answers to questions are registered in the Rules Context and as such are available to loaders. Information is then passed back to the script execution by setting dataitems of the RDO associated with the loader.

```
public class MyLoader extends Loader {
...
protected void load(final RulesParameters dtls)
    throws AppException,
    InformationalException {
```

```
...
  final String questionRef = "GroupID.QuestionID";
  String answer = null;
  // Check if answer exists.
  if (dtls.getData().containsKey(questionRef)) {
    final Object answerItem = dtls.getData().get(questionRef);
    // Check the answer is the correct type.
    if (answerItem instanceof ItemSVR_STRING) {
      answer = ((ItemSVR_STRING) answerItem).getValue(dtls);
    }
  }

...

  // Set dataitems to pass information back to IEG.
...
```

6.7 Pre-populating an IEG Script

It is possible to populate a script execution with answers provided in a previous script execution. This is achieved through API methods that execute a script, supplying answers obtained from the previous execution.

Pre-population can proceed successfully in either of two scenarios:

- Pre-populating an exact replica of the previously executed script
  Pre-population should complete correctly and all fields in the new execution should be populated correctly.
- Pre-populating a similar and compatible version of the previously executed script
  The pre-population API receives an identifier or result XML string for an execution along with a similar but still compatible script definition. As the new script definition differs in some regards, e.g. different questionIDs and group IDs, these elements cannot be populated. All other elements can be populated and execution will complete correctly.

The following API methods are available for this feature:

```
createPrepopulatedScript(executionId,
                         scriptDefinition);
```

where executionId is a long referring to the existing execution from which the answer will be taken and scriptDefinition is the new IEGScriptDefinition object to be populated.

```
createPrepopulatedScript(xml,
                         scriptDefinition);
```

where xml is a String containing the execution XML of the existing execution from which the answer will be taken and scriptDefinition is the new IEGScriptDefinition object to be populated.

6.7.1 Failures in pre-population

Where possible, IEG endeavors to pre-populate the new script; however, some changes to the script definition may result in the pre-population failing

to complete successfully. For example, if a new mandatory queston is added without a default value, it is not possible for the pre-population to proceed past the page on which the new question has been added. Another example is the case of a post-condition added to a page; if that post-condition is not satisfied, pre-population will stop at the page that contains the post-condition.

Execution will fail on points of incompatibility between the scripts (throwing the appropriate error message) until resolved by the modification of the second script. For example, if a new page is added containing mandatory questions without default answers, the following message will be displayed: 'The answers provided are not for the current page'.

The addition of pages and questions with default values is considered a trivial change to a script and should not affect pre-population from an older verison of the script definition. Changes that affect the flow of script execution are considered significant and can affect the ability to pre-populate. An example of this is the introduction of loops around existing pages.

Chapter 7

Limiting IEG Script Modification

7.1 Introduction

The ability to limit the modification of scripts or question groups may be required by the organization. Modifications to a script or question group are generally not allowed after a script has been executed. For example, case workers should not be able to modify questions at runtime. An IEG developer can define validation methods in the application that will be invoked at runtime if a modification is attempted.

7.2 Defining Validation Methods in the Application

IEG Editor actions can be limited by implementing a validation class and setting an application property to point to that class. The validation class used is defined by the application property, `curam.iegeditor.callback.class`. This application property should include the full class name including the Java package name. The validation class should implement the Java interface *curam.util.ieg.impl.IEGApplicationCallBack*. For more information on managing properties in Cúram, see the `Cúram Administration Guide`.

A sample class that implements *curam.util.ieg.impl.IEGApplicationCallBack* is provided below:

```
public class SampleValidation
  implements curam.util.ieg.impl.IEGEditorPreEditCallBack {

  public boolean canEditScript(String scriptID) {

    if (checkIfScriptHasBeenExecuted(scriptID)) {
      return false;
    } else {
      return true;
    }
  }

  public boolean canEditGroup(String groupID) {
```

```
    if (checkIfGroupIsInAnExecutedScript(groupID)) {
      return false;
    } else {
      return true;
    }
  }

}
```

7.3 Validation Processing at Runtime

When a modification is attempted at runtime, the validation methods defined in the application are invoked and the following processing occurs:

If the `curam.iegeditor.callback.class` property is set,

1. A check is performed to ascertain the existence of the class specified in the property. If the class is missing, if it fails to instantiate, or if any other error occurs, an AppException is thrown and the editing of the script of question group is disallowed.
2. If the class exists and can be instantiated, a validation method is invoked and a boolean value is returned. If this boolean value is true, editing is allowed. If it is false, an AppException is thrown and the editing of the script or question group is disallowed.

If the `curam.iegeditor.callback.class` property is not set, validation is not performed and the modification of the script or group is allowed.

Chapter 8

The IEG Editing API

8.1 Introduction

This chapter provides imformation on how to use the IEG Editing API, `curam.util.ieg.impl.IEGEditingAPI`. The Editing API provides an alternative to the IEG Editor for maintaining script and question group definitions.

8.2 Creating a Script

The `IEGEditingAPI` class provides the following methods for creating a Script:

- `createScript`: This method accepts a ScriptDetails struct as a parameter and results in the creation of a script. The ScriptDetails struct contains a *`type`* field that is ignored in this instance. Scripts that have been defined without a type will be displayed when listing scripts in the administration application. An AppException is thrown if an error occurs when creating the script.
- `createScriptWithType`: This method accepts a ScriptDetails struct as a parameter and results in the creation of a script with a designated type. Scripts that have been defined with a type will not be displayed when listing scripts in the administration application. An AppException is thrown if an error occurs when creating the script.

8.3 Modifying a Script

The `IEGEditingAPI` class provides the following method for modifying a Script:

- `modifyQuestionScript`: This method accepts a QuestionScriptDe-

tails struct as a parameter and results in the modification of a script. An AppException is thrown if an error occurs when modifying the script.

8.4 Deleting a Script

The `IEGEditingAPI` class provides the following method for deleting a Script:

- `deleteScript`: This method accepts a QuestionScriptByID struct as a parameter and results in the deletion of a script, but not the question groups associated. An AppException is thrown if an error occurs when deleting the script.

8.5 Creating a Question Group

The `IEGEditingAPI` class provides the following methods for creating a Question Group:

- `createQuestionGroup`: This method accepts a QuestionGroupDetails struct as a parameter and results in the creation of a question group. The QuestionGroupDetails struct contains a *`type`* field that is ignored in this instance. Question groups that have been defined without a type will be displayed when listing question groups in the administration application. An AppException is thrown if an error occurs when creating the question group.
- `createQuestionGroupWithType`: This method accepts a QuestionGroupDetails struct as a parameter and results in the creation of a question group with a designated type. Question groups that have been defined with a type will not be displayed when listing question groups in the administration application. An AppException is thrown if an error occurs when creating the question group.

8.6 Deleting a Question Group

The `IEGEditingAPI` class provides the following method for deleting a Question Group:

- `deleteQuestionGroupByID`: This method accepts a String containing the groupd ID as a parameter and results in the deletion of a question group if it is not associated with a script. An AppException is thrown if an error occurs when deleting the question group.

8.7 Listing Scripts

The `IEGEditingAPI` class provides the following methods for listing

scripts:

- `listQuestionScripts`: This method accepts no parameters and returns a list in the form of a QuestionScriptDetailsList containing all scripts defined without a type.
- `listQuestionScriptsByType`: This method accepts a QuestionDefinitionType as a parameter and returns a list in the form of a QuestionScriptDetailsList containing all scripts defined with the specified type.

8.8 Listing Question Groups

The `IEGEditingAPI` class provides the following method for listing question groups:

- `listQuestionGroups`: This method accepts no parameters and returns a list in the form of a QuestionGroupDetailsList containing all question groups defined without a type.
- `listQuestionGroupsByType`: This method accepts a QuestionDefinitionType as a parameter and returns a list in the form of a QuestionGroupDetailsList containing all question groups defined with the specified type.

8.9 Deep-cloning a Script

The `IEGEditingAPI` class provides the following method for deep-cloning a script:

- `deepClone`: This method accepts a DeepCloneDetails struct as a parameter and returns a QuestionScriptByID containing the details of the newly created script. This will also clone the associated question groups and subscripts. The version number provided will be appended after a $ to the question group and subscript names contained in the script. An AppException is thrown if an error occurs when deep-cloning the script. In particular, the following is checked to ensure no naming conflicts will occur: if the version number already exists in the name of a question group or subscript, it must be lower than the new number.

8.10 Listing Questions

The `IEGEditingAPI` class provides the following method for listing questions:

- `listQuestionsForScript`: This method accepts a QuestionScriptByID as a parameter and returns a list in the form of a QuestionDetailsList containing all questions defined in the specified script.

8.11 Listing Question Aliases

The `IEGEditingAPI` class provides the following method for listing question aliases:

- `listQuestionAliases`: This method accepts a QuestionAliasesByLocale as a parameter and returns a list in the form of a QuestionAliasDetailsList containing all aliases defined in the specified question.

Chapter 9

The IEG Execution Widget

9.1 Introduction

This chapter provides information on the layout of the IEG Execution Widget. It also provides the IEG developer with information on how to configure IEG Execution Widget properties. These properties are configured from three separate files. The basic layout of the IEG Execution Widget is configured from an XML file. Localizable properties such as panel names are configured from a properties file. Style properties such as panel position, color, and height are configured from a CSS file.

9.2 IEG Execution Widget Layout

The IEG Execution Widget is made up of four distinct panels. These are the tab panel, the question script panel, the question panel, and the navigation panel. The tab panel consists of a number of different views that display information relating to the execution of a script. The question script panel displays scripted text that has been defined for the current question. The question panel displays the group of questions that have been defined for the current page in a script. The navigation panel displays the controls that are used to navigate through a script.

9.2.1 Tab Panel

The tab panel consists of the tab bar and the selected tab panel. The selected tab panel displays the contents of the currently selected tab. There are five configurable tab panels. These are the pages panel, the help panel, the notes panel, the unanswered panel, and the summary panel. The following sections describe each of these panels in more detail.

Pages Panel

The pages panel displays a list of the pages that have been displayed during the current script execution. The name of each page is displayed in the order in which the pages were opened. In the case of looped pages, each individual pass of the loop will be represented on the pages panel.

Help Panel

The help panel displays help text defined for the page currently open in the question panel. The help text consists of the page name, the page help, and the question help defined using the IEG Editor. In addition, images are used to represent links to policy and legislation. These links can be defined for a page or question using the IEG Editor. The image is not displayed if a link has not been defined for a page or question.

Notes Panel

The notes panel is used to record miscellaneous information during the script execution. A recorded note is associated with the complete script execution rather than with an individual page.

Unanswered Panel

The unanswered panel is used to list questions that have been marked by the user as unanswered.

Summary Panel

The summary panel displays a list of the questions asked and the answers given during the current script execution.

9.2.2 Question Script Panel

The question script panel displays scripted text that has been defined for the current question. As the focus is moved from question to question, the question script panel is updated to display the appropriate script.

9.2.3 Question Panel

The question panel can be broken down into three areas. These are the page title bar, the page notes area, and the questions and answers area. The page title bar displays the page name. The page notes area displays any notes that are defined for the page. The questions and answers area displays the questions asked and the answers given on the page.

9.2.4 Navigation Panel

The navigation panel displays the controls that are used to navigate through a script. The controls consist of the exit, next, and previous buttons. The exit

button is used to terminate the current script execution. The previous button is used to return the user to the page that was displayed immediately before the current page. The next button is used to submit the data entered on the current page and to then open the next page.

9.3 XML Configuration File

A configuration file called `IEGPlayer.xml`defines whether the `tab-panel` element and its child elements (`previous-pages-panel`, `help-panel`, `notes-panel`, `unanswered-panel`, `summary-panel`) will be displayed at runtime. It also defines the `finish-page` element that will be displayed when the exit button is clicked or the last question page has been answered.

The configuration file must be saved in a component. If multiple versions of this file exist in separate components, the version from the highest priority component will be used.

The following is an example of a configuration file.

```
<!-- Example Configuration File -->

<ieg-player-config>

  <finish-page page-id="IEGFinish"
               execution-id-param-name="executionID"
               case-id-param-name="caseID"
               participant-id-param-name="participantID"  />

  <tab-panel visible="true">
    <previous-pages-panel visible="true" />
    <help-panel visible="true" />
    <notes-panel visible="true" />
    <unanswered-panel visible="true" />
    <summary-panel visible="true" />
  </tab-panel>

</ieg-player-config>
```

The `tab-panel` element and its child elements are all optional. If the `visible` property on an element is set to true, the associated panel will be displayed at execution. The `visible` property defaults to true on each element. Note that if the `visible` property of the `tab-panel` element is set to false, the entire left-hand panel is hidden and the child element properties will have no effect.

The following table lists the properties that can be configured from the XML file.

Property Name	Description
`tab-panel`	Defines if the tab panel is displayed.
`previous-pages-panel`	Defines if the pages panel is displayed.
`help-panel`	Defines if the help panel is displayed.
`notes-panel`	Defines if the notes panel is dis-

Property Name	Description
	played.
unanswered-panel	Defines if the unanswered panel is displayed.
summary-panel	Defines if the summary panel is displayed.
page-id	The UIM page ID of the finish page.
execution-id-param-name	Defines the name of the page parameter containing the script execution ID. The execution ID will be passed on to the finish page.
case-id-param-name	Unique identifier of the associated case. This defines the name of the page parameter containing the case ID. The case ID will be passed on to the finish page. This property is optional.
participant-id-param-name	Unique identifier of the associated participant. This defines the name of the page parameter containing the participant ID. The participant ID will be passed on to the finish page. This property is optional.

Table 9.1 XML File Properties

9.4 Properties File for Text Resources

The IEG Execution Widget contains various text properties which can be localized. These properties are contained in a properties file which can be found in the CuramCDEJ/sample/JavaSource/curam/omega3/i18n/IEGPlayer.properties.sample file. This file displays comments and default values for each property.

To change the default values for a property, the above file must be copied to the JavaSource/curam/omega3/i18n/ folder of the client project and the .sample extension must be removed. The properties listed in the following table can then be configured.

Property Name	Description
previous.pages.button.label	The name associated with the previous pages panel. This name is also used as the label of the button used to display the panel.

Property Name	Description
previous.pages.button.alt	Alternative text for the button used to display the previous pages panel.
help.button.label	The name associated with the help panel. This is also used as the label of the button used to display the panel.
help.button.alt	Alternate text for the button used to display the help panel.
notes.button.label	The name associated with the notes panel. This is also used as the label of the button used to display the panel.
notes.button.alt	Alternative text for the button used to display the notes panel.
notes.input.alt	Alternative text for the text input control used to enter notes.
unanswered.button.label	The name associated with the unanswered questions panel. This is also used as the label of the button used to display the panel.
unanswered.button.alt	Alternative text for the button used to display the unanswered questions panel.
summary.button.label	The name associated with the summary panel. This is also used as the label of the button used to display the panel.
summary.button.alt	Alternative text for the button used to display the summary panel.

Table 9.2 Text Resource File Properties

9.5 CSS File for Style Properties

The IEG Execution Widget contains several style properties that can be configured using CSS. CSS style properties control aspects such as font, height, background color, etc.

The default style properties for the IEG Execution Widget can be accessed in the `CuramCDEJ/lib/curam/web/css/curam_ieg.css` file. These properties generally control fonts and images, but the developer can customize the content of these properties with any currently supported CSS feature.

The CSS file is commented to provide the IEG developer with information on each property and on the default content of each property.

To customize the default style properties, a file with a suffix `_ieg.css` must be created and placed in a component, e.g., the `components/cus-`

tom/custom_ieg.css component. More than one custom CSS file can be created as long as they are named differently. For example, a developer may chose to customize each panel separately using several CSS files.

Note that the CuramCDEJ/lib/curam/web/css/curam_ieg.css file does not have to be duplicated in its entirety; only the properties that will be changed must be added to the custom file. The properties in the new file must have the same name as those in the original file in order to be picked up by the build process. The content of the properties can, however, be edited freely.

More than one property may have to be configured to control certain aspects of the IEG Execution Widget. For example, to control the relative positions of the tab panel and the question panel, both the .tab-panel and .detail-panel properties must be configured.

The following table lists the properties that can be configured using the CSS file.

CSS Property Name(s)	Description
.tab-panel	Used to control the tab panel style.
.tab-panel and .detail-panel	Used to control the relative positions of the tab panel and question panel.
div#question-script-panel	Used to control the question script panel style.
div#question-panel	Used to control the question panel style. Also used to control the question panel's position. The question panel can be displayed on either the top or the bottom of the page.
div#navigation-panel	Used to control the navigation panel style. Also used to control the navigation panel's position. The navigation panel can be displayed on either the top or the bottom of the page.
td#exit-button input	Used to control whether or not the exit button is displayed.
td#nav-buttons input	Used to control whether or not the next and previous buttons are displayed. Note that these two buttons cannot be configured separately.
.page-notes	Used to control the page notes style and to define whether or not page notes are displayed. Page notes are always displayed immediately above the question & answer area.
div#question-panel and td#page-title	Used to control whether or not the page title bar is visible or not.

CSS Property Name(s)	Description
a.page	Used to control the style of the icon associated with a page. This is used in any of the panels that display a page name. It also defines whether or not the icon is displayed.
a#previous-pages-panel-tab	Used to control the style of the pages panel icon that is displayed in the tab bar. It also defines whether or not the icon is displayed.
a#help-panel-tab	Used to control the style of the help panel icon that is displayed in the tab bar. It also defines whether or not the icon is displayed.
a#notes-panel-tab	Used to control the style of the notes panel icon that is displayed in the tab bar. It also defines whether or not the icon is displayed.
.question-unans	Used to control whether or not the unanswered check box is displayed.
a#unanswered-questions-panel-tab	Used to control the style of the unanswered panel icon that is displayed in the tab bar. It also defines whether or not the icon is displayed.
a#summary-panel-tab	Used to control the style of the summary panel icon that is displayed in the tab bar. It also defines whether or not the icon is displayed.
a.policy	Used to control the style of the policy image for a policy link on a page or question. It also defines whether or not the image is displayed.
a.legislation	Used to control the style of the legislation image for a legislation link on a page or question. It also defines whether or not the image is displayed.
td#page-title-help	Used to control the style of the help link Image. It also defines whether or not the image is displayed.

Table 9.3 CSS File Properties

9.6 IEG Test Player

The IEG Test Player has been provided as a mechanism to conveniently test question scripts defined in the IEG Editor without the need for any application development. However, it is not possible to test scripts that require preinitialized RDOs using the Test Player.

The IEG Test Player is invoked by selecting the 'Simulation' link from navigation menu in the Cúram Application. The user is presented with a list of scripts present in the database. Upon choosing one to run (and confirming that choice), the IEG Execution Widget appears on screen and execution commences.

When execution is finished, the user is given the choice of removing the script execution object from the database or retaining it. Scripts will persist in the database until they are explicitly removed. It is the responsibility of the application developer to implement a mechanism to maintain a link between the execution ID and information to help the user identify a particular test script execution.

Chapter 10

Using Gathered Evidence

10.1 Introduction

After a user has completed a script execution, the information gathered is stored in the database as a serialized Java object. The IEG Execution API converts this into XML so that it can be used for further processing. After the XML is retrieved, the API is used to remove the IEG execution information from the database. The API can then extract relevant elements from the XML data for use in additional processing. Alternatively, the developer can store the XML data to another file or database as required by the organization.

It is recommended that the API be used to extract relevant elements from the XML data rather than directly manipulating the XML. This will help reduce any impact from future possible changes to the XML structure.

10.2 Retrieving XML Data

The IEG Execution API converts the data stored from a script execution from a serialized Java object to XML. The XML data can then be accessed as a String using the `getResult()` method from within the `IEGRuntime` class. This method returns a `curam.util.ieg.struct.ScriptResult` struct which in turn contains the result String in its *`scriptResult`* member variable.

If IEG execution is interrupted, the partial results of the script (not including the information on the interrupted page) will be returned as XML data in the same way as for a completed script execution.

The following example shows how to retrieve the XML data gathered during the script execution:

```
// retrieve the XML data gathered during the script execution
```

```
import curam.util.ieg.struct.ScriptResult;can be stored
import curam.util.ieg.struct.ExecutionIdentifier;
...
IEGRuntime iegRuntime;
ExecutionIdentifier executionIdentifier;
...
// Instantiate iegRuntime and populate executionIdentifier.
...
ScriptResult scriptResult = iegRuntime.getResult(executionId);
String xmlResult = scriptResult.scriptResult;
```

10.3 Removing Data from the Database

After the data gathered during execution has been converted to XML data, it can be removed from the database using the `removeScript()` method on the `IEGRuntime` class.

The following example shows how data is removed from the database:

```
// remove a execution data from the DB

import curam.util.ieg.struct.ExecutionIdentifier;
...
IEGRuntime iegRuntime;
ExecutionIdentifier executionIdentifier;
...
// Populate iegRuntime and executionIdentifier.
...
ScriptResult scriptResult = iegRuntime.removeScript(executionId);
```

10.4 Extracting XML Data

After the XML data is retrieved, it can be examined and the relevant elements (i.e., page results or question results) can be extracted using the class `curam.util.ieg.impl.IEGScriptResult`. In order to create an instance of this class, the XML data is passed from the `iegRuntime.getResult` method to the constructor of the `curam.util.ieg.impl.IEGResultAccessor` class.

The following example shows how to retrieve the XML data gathered during the script:

```
// retrieve the XML data gathered during the script execution

xmlString = iegRuntime.getResult(executionId);
IEGScriptResult scriptResult =
    IEGResultAccessor.getScriptResult(xmlString);
```

To access question page results, the `getPageResult()` method is called with the question group ID associated with the page. Within the `curam.util.ieg.impl.QuestionPageResult` class, any child question pages can also be accessed by calling the `getPageResult()` method.

To access question results, the `getResult()` method is called with the question ID associated with the question. This returns a

`curam.util.ieg.impl.QuestionResult` object which contains data such as the default answer (by calling the `getDefaultAnswer()` method), the loop index (by calling the `getLoopIndex()` method), etc.

The following example shows how to access a page result object:

```
// get a page result object

QuestionPageResult pageResult = scriptResult.getPageResult("QP1");

// child page within page QP1
pageResult = pageResult.getPageResult("QP2");
```

If a child page has a loopsize of greater than 0, then the specific loopindex must be included:

```
// get a page result object using the loop index

pageResult = pageResult.getPageResult("QP3", 0);

// The question result QP3_Q1 of within the question page QP2
QuestionResult result = pageResult.getResult("QP3_Q1");
assertEquals("Q4 Question", result.getDefaultAnswer());
assertEquals(1, result.getLoopIndex());

// Iterate through each of the question pages retrieving the
// question results.
for (int i = 0; i < pageResult.getLoopSize(); i++) {getLoopIndex()
  result = pageResult.getResult("QP3", i);
}
```

10.5 Storing XML Data for Future Use

A developer can store XML data to any file or database. This XML data can subsequently be accessed as required by the organization.

Chapter 11

Import and Export of IEG Definitions

11.1 Introduction

IEG script and question group definitions are stored in the database. It is possible to import IEG definitions from the file system to the database. Conversely, it is also possible to export definitions from the database to the file system in order to facilitate version control for example.

A number of the import targets rely in the script and question group definitions being stored in a particluar directory. For example, when importing from a component, definitions are imported from the `ieg` subdirectory of the specified component. Similarly, the commands that deal with exporting to a particualr component will place the script and group definitions in the `ieg` subdirectory of the specified component.

The extensions of files containing the definitions are also significant for targets that import several files. Script definitions should be stored in files with the extension `'.sx'` and question group definitions should be stored in files with the extension `'.gx'`.

The command-line statements used for import and export functionality (see also the `Cúram Server Developer's Guide`) are:

- exportiegscript
- exportiegscripttodir
- exportfulliegscript
- exportfulliegscripttodir
- exportquestiongroup
- importieg
- importiegscript

- importiegcomponent
- importiegsubdirs
- importquestiongroup

11.2 IEG Import Commands

IEG definitions can be imported from the file system to the database in a number of ways. The following table lists the various import commands and their parameters:

Command	Description	Parameters
importieg	Imports script and question group definitions from a specified directory.	directory (mandatory), overwrite
importiegscript	Imports a script definition from the specified file.	ieg.file (mandatory), overwrite
importiegcomponent	Imports script and question group definitions from the ieg subdirectory of a specified component.	component (mandatory), overwrite
importiegsubdirs	Imports script and question group definitions from the ieg subdirectories of all the subdirectories of a specified directory.	directory (mandatory), overwrite
importquestiongroup	Imports a question group definition from the specified file.	ieg.file (mandatory), overwrite

Table 11.1 IEG Import Commands

Each of these targets allows an optional overwrite parameter, which defaults to 'false' and indicates whether any existing definitions with the same ID as the import definitions should be overwritten.

11.3 IEG Export Commands

An IEG Script can be created in the editor and then exported to the file system using one of the commands listed below. This allows definitions created in the IEG editor to be placed under source control.

Exporting definitions causes files to be created in the target directory with separate files for each definition. The name of the files corresponds to the ID of the definition it contains with the extension `'.sx'` for scripts and `'.gx'` for question groups.

Command	Description	Parameters
exportiegscript	Exports an IEG script definition to the file system, given a script ID and the component to which the script should be exported. Any question group definitions associated with the script will not be exported.	scriptid (mandatory), component (mandatory)
exportiegscripttodir	Exports an IEG script definition to the file system, given a script ID and the full path of the directory to which the script should be exported. Any question group definitions associated with the script will not be exported.	scriptid (mandatory), exportdirectory (mandatory)
exportfuliegscript	Exports an IEG script definition and all its associated question group definitions to the file system, given a script ID the and component to which the script should be exported.	scriptid (mandatory), component (mandatory)
exportfuliegscripttodir	Exports an IEG script definition and all its associated question group definitions to the file system, given a script ID the and the full path of the directory to which the script should be exported.	scriptid (mandatory), exportdirectory (mandatory)
exportquestiongroup	Exports an IEG question group definition to the file system.	groupid (mandatory), component (mandatory)

Table 11.2 IEG Export Commands

Chapter 12

Adding IEG Administration Pages

12.1 Introduction

The version of IEG that this document covers has been superseded and has been removed from the administration application. This version of IEG is in maintenance mode and the new version of IEG is now the preferred technology for new development. Please refer to the Authoring Scripts using Intelligent Evidence Gathering(IEG) Developer's Guide for information on script design, development, and execution using the new technology. Existing use of IEG within the application is unaffected by this change and it is only the administration function that has been removed. This chapter covers adding the administrative function of this superseded version of IEG back into the application.

Please see the Application Configuration chapter of the Cúram Web Client Reference Manual for information on creating and an explanation of Sections, Section Shortcuts and Tabs.

12.2 IEG Section

An appropriate Section should be indentified to which IEG tabs can be inserted by adding entries similar to the following:

```
<sc:tab id="SUPERSEDEDIEG"/>
<sc:tab id="IEGScript"/>
```

Example 12.1 Section.sec

An entry similar to the following can then be added to the Shortcuts for the Section:

```
<sc:node type="group"
         title="SUPERSEDED.IEG.Title"
         id="SUPERSEDEDIEG ">
```

```
    <sc:node type="leaf"
          id="SUPERSEDEDListIEG"
          page-id="IEG_ListQuestionScripts"
          title="SUPERSEDED.ListIEG.Title">
   </sc:node>
</sc:node>
```

Example 12.2 Shortcuts.ssp

Corresponding properties can be added to the Shortcuts properties file for the Section:

```
SUPERSEDED.IEG.Title=SUPERSEDED Intelligent Evidence Gathering
SUPERSEDED.ListIEG.Title=IEG Scripts
```

Example 12.3 Shortcuts.properties

12.3 IEG Tabs

Two tab configurations are required to be added. One tab is used to list the scripts and the other is used for simulating script execution. The tab configuration files should have corresponding properties files. The contents of the tab configurations and properties will be similar to the following:

```
<tc:tab-config xmlns:tc=
     "http://www.curamsoftware.com/curam/util/client/tab-config"
  id="SUPERSEDEDIEG">
  <tc:menu id="IEGMenu"/>
  <tc:context tab-name="Details.Name.IEG"
              tab-title="Details.Title.IEG"/>
  <tc:navigation page-id="IEG_ListQuestionScripts"/>
</tc:tab-config>
```

Example 12.4 SUPERSEDEDIEG.tab

```
Details.Name.IEG=IEG
Details.Title.IEG=IEG
```

Example 12.5 SUPERSEDEDIEG.properties

```
<tc:tab-config xmlns:tc=
     "http://www.curamsoftware.com/curam/util/client/tab-config"
  id="IEGScript">
  <tc:page-param name="questionScriptIDParam"/>
  <tc:menu id="SimpleAdminMenu"/>
  <tc:context tab-name="Details.Name.IEGScript"
              tab-title="Details.Title.IEGScript"/>
  <tc:navigation page-id="IEG_TreeWindow"/>
</tc:tab-config>
```

Example 12.6 IEGScript.tab

```
Details.Name.IEGScript=IEG Script
```

```
Details.Title.IEGScript=IEG Script
```

Example 12.7 IEGScript.properties

12.4 IEG Menu

The tab to list the IEG script also requires a menu to be defined so that question groups may be created, script changes can be published and script executions can be simulated. The menu configuration should be similar to the following:

```
<mc:menu-bar xmlns:mc=
    "http://www.curamsoftware.com/curam/util/client/menubar-config"
  id="IEGMenu">
  <mc:submenu id="NEW"
              title="Submenu.Title.New"
              tooltip="Submenu.Toolip.New">
    <mc:menu-item  id="QuestionGroup"
      page-id="IEG_InsertQuestionGroup"
      title="MenuItem.Title.QuestionGroup"
      tooltip="MenuItem.Tooltip.QuestionGroup" />
  </mc:submenu>
  <mc:menu-item  id="PublishScripts"
    page-id="IEG_PublishChanges"
    title="MenuItem.Title.PublishScripts"
    tooltip="MenuItem.Tooltip.PublishScripts" />
  <mc:menu-item  id="SimulateScriptRun"
    page-id="IEG_ListQuestionScriptsPlayerTest"
    title="MenuItem.Title.SimulateScriptRun"
    tooltip="MenuItem.Tooltip.SimulateScriptRun" />
</mc:menu-bar>
```

Example 12.8 IEGMenu.mnu

A corresponding properties file should be added for the menu containing properties similar to the following:

```
Submenu.Title.New=New
Submenu.Toolip.New=New

MenuItem.Title.QuestionGroup=Question Group
MenuItem.Tooltip.QuestionGroup=Question Group

MenuItem.Title.PublishScripts=Publish Scripts
MenuItem.Tooltip.PublishScripts=Publish Scripts

MenuItem.Title.SimulateScriptRun=Simulate Script Run
MenuItem.Tooltip.SimulateScriptRun=Simulate Script Run
```

Example 12.9 IEGMenu.properties

12.5 Inserting Tab Configuration

Once the changes have been made to the Section, Section Shortcuts and properties files and the Tab Menu and properties files have been created the

configuration should be loaded into the database by invoking the `insert-tabconfiguration` command line target as described in `Cúram Web Client Reference Manual`.

Appendix A

Operations Supported for IEG Expressions

A.1 Introduction

This appendix provides information on the operations that are supported for IEG expressions. It also provides information on the bracketing of terms and on operator precedence.

A.2 Bracketing of Terms

The bracketing of terms can have a significant impact on the result of a calculation. The behavior is as normal for mathematical operations, but the effects of brackets can be combined with operator precedence (see below) and may add complexity to an expression. Any operation that should be carried out in advance of another operation should be bracketed, e.g., 5 * (3/4) = 3.75.

A.3 Operator Precedence

The precedence of operators is as defined for the Java programming language. The operators in the following table are listed in order of precedence:

Operator	Associatively	Type
()	left to right	parentheses
* /	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equities

Table A.1 Operator Precedence

A.4 Data Types and Supported Operations

The operations that are explicitly supported between the data types are detailed in the table below.

It is possible to perform operations between the data types not listed in the table if the underlying data type of an attribute can be converted into one of the types for which an operation is supported.

For example, the addition of SVR_INT8 and SVR_MONEY is possible, because SVR_INT8 is converted into SVR_DOUBLE and the addition of SVR_DOUBLE and SVR_MONEY is supported.

It is possible to add or subtract integers from dates. Integers represent the number of days to be added or subtracted.

The first parameter type	The second parameter type	Operations supported	Result type
SVR_STRING	SVR_STRING	==, !=	SVR_BOOLEAN
SVR_CHAR	SVR_CHAR	==, !=	SVR_BOOLEAN
SVR_MONEY	SVR_MONEY	==, !=, <, >, <=, >=	SVR_BOOLEAN
SVR_MONEY	SVR_DOUBLE	==, !=, <, >, <=, >=	SVR_BOOLEAN
SVR_DOUBLE	SVR_MONEY	==, !=, <, >, <=, >=	SVR_BOOLEAN
SVR_DOUBLE	SVR_DOUBLE	==, !=, <, >, <=, >=	SVR_BOOLEAN
SVR_DATE	SVR_DATE	==, !=, <, >, <=, >=	SVR_BOOLEAN
SVR_DATE	SVR_DATETIME	==, !=, <, >, <=, >=	SVR_BOOLEAN
SVR_DATETIME	SVR_DATETIME	==, !=, <, >, <=, >=	SVR_BOOLEAN
SVR_DATETIME	SVR_DATE	==, !=, <, >, <=, >=	SVR_BOOLEAN
SVR_MONEY	SVR_MONEY	+, -, /, *	SVR_DOUBLE
SVR_MONEY	SVR_DOUBLE	+, -, /, *	SVR_DOUBLE
SVR_DOUBLE	SVR_MONEY	+, -, /, *	SVR_DOUBLE
SVR_DOUBLE	SVR_DOUBLE	+, -, /, *	SVR_DOUBLE
SVR_FLOAT	SVR_FLOAT	+, -, /, *	SVR_DOUBLE
SVR_INT8	SVR_INT8	+, -, /, *	SVR_INT32

The first parameter type	The second parameter type	Operations supported	Result type
SVR_INT16	SVR_INT16	+, -, /, *	SVR_INT32
SVR_INT32	SVR_INT32	+, -, /, *	SVR_INT32
SVR_INT64	SVR_INT64	+, -, /, *	SVR_INT64
SVR_DATE	SVR_INT32	+, -	SVR_DATE

Table A.2 Data Types and Supported Operations

Appendix B

Answer Data Types

B.1 Available Answer Data Types

The following table provides a list of answer data types currently supported for use in IEG.

Domain Definition	Answer Data Type	Description
SVR_BOOLEAN	Check box	This type should be used for questions requiring only “yes”/“no”, “true”/“false” as an answer and corresponds to the primitive java type `boolean`.
SVR_CHAR	Text entry box	This type should be used for questions requiring a single character as an answer and corresponds to the primitive java type `char`.
SVR_DATE	Date selection box	This type should be used for questions requiring a date as an answer. For example, a date of birth. This type corresponds to java class `curam.util.type.Date`.
SVR_DOUBLE	Text entry box	This type should be used for questions requiring a floating point

Domain Definition	Answer Data Type	Description
		number as an answer. For example, a housing area. This type corresponds to the primitive java type `double`.
SVR_INT8	Text entry box	This type should be used for questions requiring an integer answer. This type corresponds to the primitive java type `byte`.
SVR_INT16	Text entry box	This type should be used for questions requiring an integer answer. This type corresponds to the primitive java type `short`.
SVR_INT32	Text entry box	This type should be used for questions requiring an integer answer. This type corresponds to the primitive java type `int`. This should be used as the default type for integers.
SVR_INT64	Text entry box	This type should be used for questions requiring an integer answer. This type orresponds to the primitive java type `long`.
SVR_MONEY	Text entry box	This type is a fixed point numeric value with two decimal places and should be used for questions requiring a money amount as an answer. For example, the monthly income amount. This type corresponds to java class `curam.util.type.Money`.
SVR_STRING	Text entry box	This type should be used for questions re-

Domain Definition	Answer Data Type	Description
		quiring text as an answer. For example, a person's name. This type corresponds to the java class `java.lang.String`.
CODETABLE_CODE	Single-select or multi-select list or drop-down list	This type should be used for questions requiring the user to choose one or more answers from a given list. For example, a country of residence.

Table B.1 IEG Answer Data Types

Note that Code Table Hierarchies are not supported in IEG.

B.2 Defining Additional Answer Data Types

Additional answer data types can be defined for use in IEG. To define additional types, a domain definition must be added to the “QuestionTypes” code table. The domain definition can be new or can already exist within the Cúram development environment. For information on how to specify a new domain definition, see the Cúram Modeling Guide.

The domain definition can be added to the code table at development time or as part of application administration. At development time, the code table file CT_QuestionTypes.ctx can be modified to add the domain definition (see the Cúram Server Modeling Guide). As part of application administration, a new domain definition can be added to the list of domain definitions stored as code table items (see the Cúram Administration Guide).

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing

Legal and Intellectual Property Law.

IBM Japan Ltd.

1623-14, Shimotsuruma, Yamato-shi

Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typograph-

ical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you. Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

Dept F6, Bldg 1

294 Route 100

Somers NY 10589-3216

U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources.

IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products

should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs.

© Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trade-