ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUPV049

THE IBEX SCRIPT GENERATOR

James King, Jack Harper, Thomas Löhnert, Aaron Long, Dominic Oram,
STFC/RAL/ISIS, Chilton, Didcot, Oxon, UK

Abstract

Experiment scripting is a key element of maximising utilisation of beam time at the ISIS Neutron and Muon Source, but can be prone to typing and logic errors. The IBEX Script Generator enables collaboration between instrument scientists and users to remove the need to write a script for many experiments, so improving reliability and control.

For maximum applicability, the script generator needs to be easily configurable which is achieved by instrument scientists creating script definitions. Script definitions are Python classes that contain the parameters a user can fill in for each action in the script, and functions to execute, validate and provide a time estimation for each action.

A user organises a table of actions and fills in their values to create their experiment, these action values are validated in real time. With a valid table of actions a user can generate a Python script which can be executed via the IBEX scripting tools.

A key requirement of the script generator is for it to integrate the pre-existing Java based IBEX client. Py4J is used as a bridge between Java and the Python script definitions. An iterative, user-focused approach has been employed with quality assurance techniques such as user interface (UI) testing to achieve a behaviour-driven development workflow.

Further planned development includes dynamically controlling the execution and values of actions whilst the script is running, action iteration and user experience improvement.

INTRODUCTION

IBEX [1] is an EPICS based control system developed and used at the ISIS Neutron and Muon Source to control beamline equipment and experiments. A key feature of IBEX is the Python-based control and scripting library known as genie python [2]. This library provides functions to control experiments in an automated manner, beginning and ending data collection, writing to EPICS process variables (PVs) to control equipment amongst a host of other functionality.

Scripting is an integral part of how ISIS runs experiments, both in IBEX and the previous control system SECI. Giving users the power to write scripts allows them to control and automate experiments in a very customisable and expressive fashion. It also helps improve the reproducibility of experiments and enables maximum utilisation of beam time.

However, there are a few pitfalls when it comes to using scripting extensively. To script an experiment a user needs to understand how to code in Python, this places a steep learning curve ahead of users who have little or no experience in coding. This learning curve distracts user focus away from the science of an experiment. Furthermore, even if a user is proficient in coding they are not familiar with genie python and IBEX, so they must learn a new complex set of commands and logic in order to correctly run their experiment. For users that are familiar with the environment it is certainly easier, but writing scripts is still prone to logic errors and mistyping of commands.

There are attempts to mitigate many of these issues within genie python, such as checking of scripts for programming errors at load time, providing a reduced command set for specific instruments and providing autocomplete for PVs of interest. One of the main roles of the script generator is to mitigate the pitfalls surrounding scripting as well as enhancing the experience of a user at the facility by enabling them to focus on the experiment.

There are a number of script generators in use at ISIS with varying degrees of functionality - some that work with the previous control system SECI. The aim of the IBEX script generator, produced by the experiment controls group [3], is to provide a unified tool – taking inspiration from other script generators currently in use at ISIS – for generating scripts without having to write code for specific experiments.

WORKFLOWS AND FUNCTIONALITY

Many experiments at ISIS follow a regular pattern. Users and scientists often customise previous scripts that follow a similar pattern to the experiment they are creating. The script generator targets this workflow by allowing instrument scientists to create script definitions – which define the parameters an experiment can take and the logic to run the experiment – and users to input experiment parameters. Using these two inputs the script generator can then generate a script (see Fig. 1).

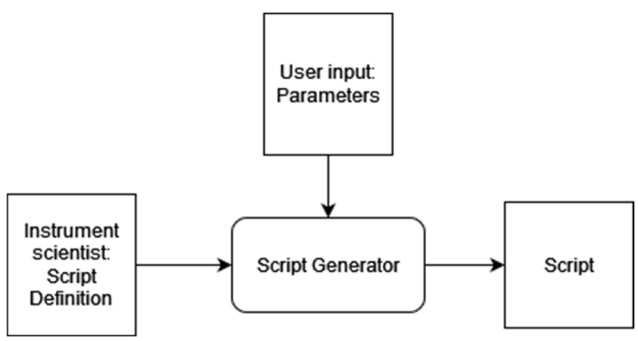

Figure 1: Script Generator inputs and outputs.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUPV049

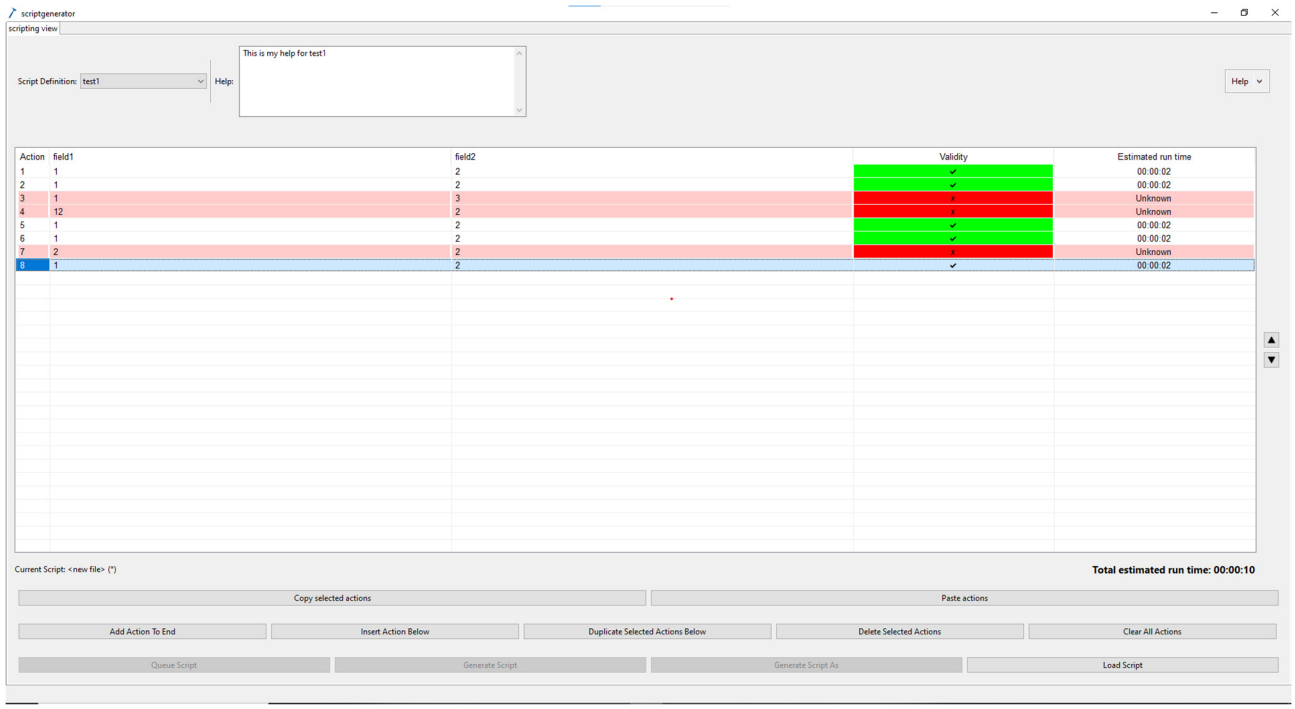

Figure 2: Script Generator User Interface.

To create their experiment the user must select the script definition (see Fig. 3) that contains the logic to run it and input their parameters into a table (see Fig. 2). In the table rows are individual actions for the script to run and columns are the experiment parameters to run each action with.

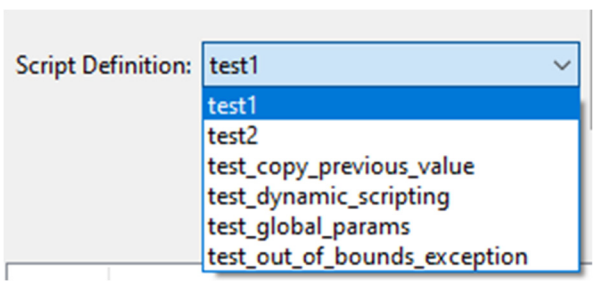

Figure 3: Script Definition selector.

There are a number of gestures (user actions) to control the order and values of actions. These include copy/paste, insert, append, delete and duplicate gestures, as well as the ability to tab between cells and reorder actions.

A user can generate a script and save it to file. This script can be executed in the IBEX scripting console. Alternatively a script can be previewed and sent to the script server [4] – a server that runs queues of Python scripts separately to the scripting console.

One requirement of the script generator is to be able to reload a script back into the script generator and edit it. To achieve this, when generating a script the parameters entered by the user are also saved to a separate json file. This file can then be reloaded into the script generator for use at a later point.

Script generator parameters are validated in real time. The validation aims to ensure incorrect values and mistyping do not end in generated scripts executing incorrectly or failing to run.

Scientists provide a validation function in a script definition. This function is arbitrary Python code which validates individual actions' parameter values. If invalid this function provides a string containing a message to display to the user as a tooltip when hovering over the action. The validity of actions are displayed through colouring an actions row.

Another feature which adds an output (read-only) column is time estimation. A script definition defines a time estimation function, which takes an actions parameter values and returns a number of seconds that this action will take to run. The time estimation column then displays this time to run in hours, minutes and seconds for each action. A total time to run is also calculated and displayed.

There are two types of parameters in the script generator, action parameters and global parameters. The previously mentioned parameters that are contained in the table are action parameters and only apply to one action. Global parameters are only set once for the whole script. For example, a script definition may use a global parameter to select a temperature controller for the whole experiment, and then individual actions may have a parameter to specify the temperature to set on that temperature controller for that action.

Global parameters are also validated individually, and can be used in the validation, time estimation and execution functions for individual actions (see Fig. 4). All these features are subject to refinement and additional functionality. This future work is discussed in the future work section.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUPV049

Figure 4: Global Parameters.

ARCHITECTURE

The IBEX control system uses a client-server architecture. The client provides a variety of perspectives for instrument control including the scripting console and script server. The client is developed using Java and the Eclipse RCP framework [5] and as such follows object-oriented design patterns to form the software architecture [6].

As there is a requirement to include the script generator as part of the IBEX client, it has also been written in Java using the Eclipse RCP framework.

Python was chosen as the scripting language for IBEX because it is an easy to learn language and is widely used in the sciences. Because Python is used for IBEX scripting, scientists have experience writing Python scripts, and the genie_python library has functions for instrument control, it makes sense for script definitions to also be written in Python.

Py4J [7] is a Python and Java library that enables code from each language to access objects from the other language. We are utilising Py4J as a bridge between the Java model and script definitions.

The architecture follows the Model-View-ViewModel (MVVM) architectural pattern [8] (see Fig. 5). The Java model talks via the Py4J bridge to the script definitions.

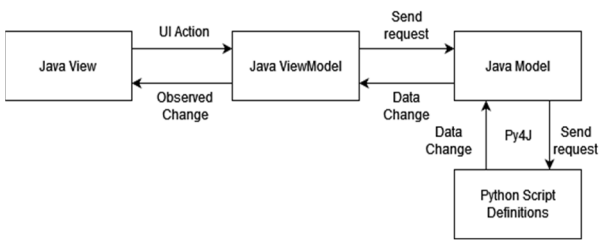

Figure 5: The Script Generator Architecture.

Python is the main language we target to generate scripts for. However, there is a requirement to be able to add new languages to generate scripts in. We have made the generation mechanism extensible using the strategy design pattern (see Fig. 7). This pattern will allow us to add extra generator languages by extending the AbstractGenerator class.

The PythonInterface class that a GeneratorPython object makes calls to is a façade on the communications over the Py4J bridge. The Python side is made up of a number of script definitions provided by instrument scientists and a number of classes providing helper methods to facilitate communication across the Py4J bridge (see Fig. 6).

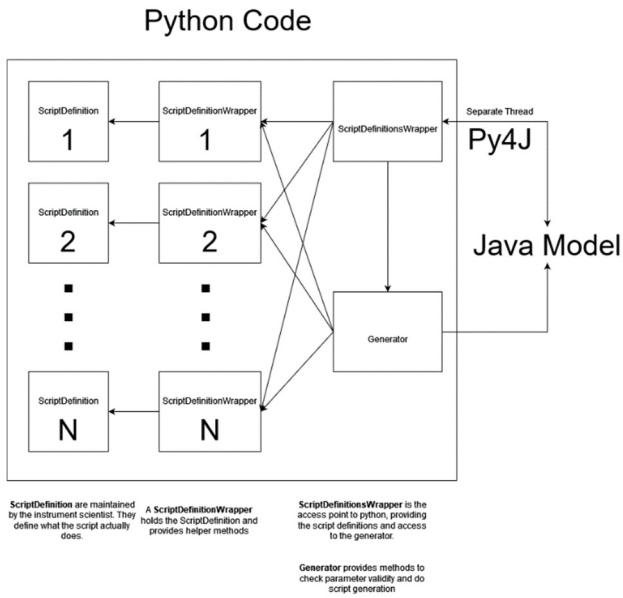

Figure 6: Python Utilities to access Script Definitions.

To compensate for the brittle and slow communication over the bridge, all communications are handled in separate threads via the Java CompletableFuture mechanism [9]. The CompletableFuture mechanism avoids blocking on the UI thread, keeping the UI responsive. However, it does mean we cannot simply set and get values from the Python side. We have thus had to develop a mechanism where the Java Model listens for the CompletableFutures to complete and passes these values back up through chain of listeners to the ViewModel from the Model.

QUALITY ASSURANCE

There are 2 forms of automated testing used to ensure the quality and behaviour of the script generator: unit testing, and system UI testing. Unit testing is carried out in Java using the JUnit framework [10] and in Python unittest [11] is used.

Both the JUnit and unittest tests are run as part of the build process, which is regularly executed on a Jenkins [12] continuous integration pipeline.

Our system UI testing is run using a separate continuous integration pipeline. These tests are written and executed with the Squish UI testing tool [13]. Squish supports a Behaviour-Driven Development (BDD) [14] approach. Tests can be laid out in the Gherkin [15] ubiquitous language. These tests reference steps that are defined as functions which carry out actions such as press buttons, fill in table cells and verify the presence and state of UI elements.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUPV049

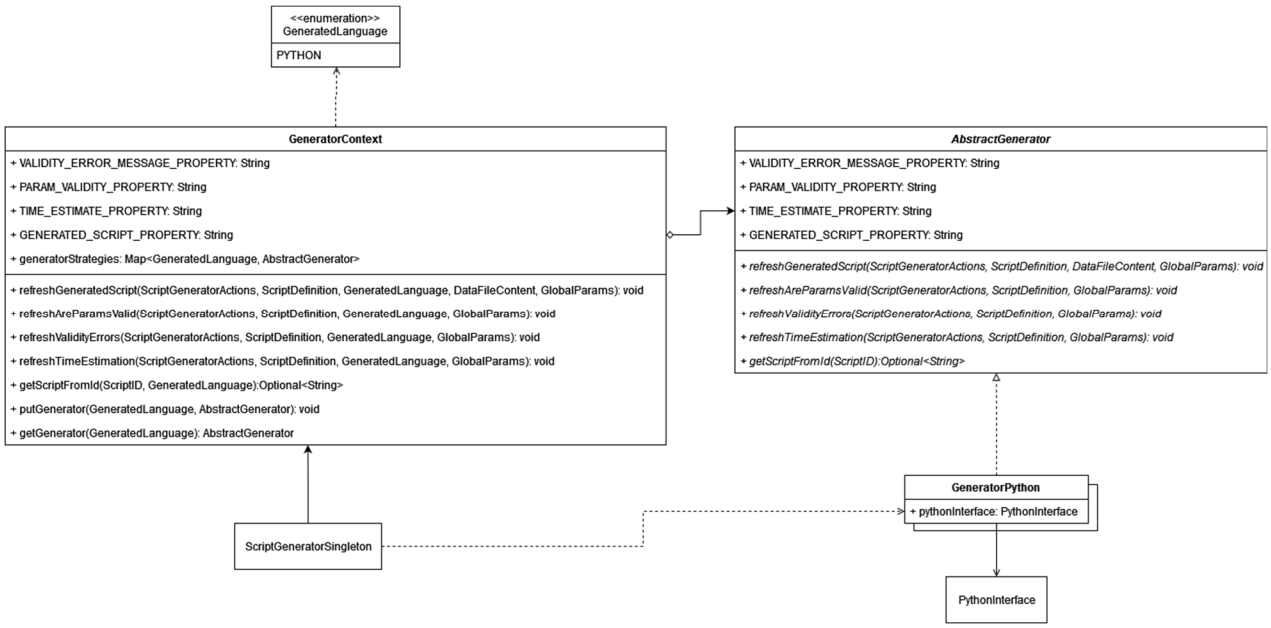

Figure 7: Strategy pattern used for generator extensibility.

We have used the Squish BDD tools [16] to describe behaviours of the script generator and verify their presence. Our suite of UI tests have caught regressions and deviations in behaviour prior to releases on multiple occasions. For example, when duplicating an action and changing the contents of one of the actions, these changes were also visible to the test on the other action revealing a shallow copy issue on the duplication mechanism.

During the release creation process we also carry out some manual system tests. These manual tests include smoke tests and tests that are difficult or impossible to automate using our current tools.

Further to testing, all code for the script generator is subject to review. These reviews help to minimise mistakes, share knowledge and techniques. Reviews also ensure consistency is practiced throughout the system, and requirements and quality standards are met. These reviews are an integral part of our quality assurance and often lead to reworks which drastically improve software quality and functionality, and the underlying code.

We run the static code analysis tool Checkstyle [17] on our Java code to identify programming flaws and enforce consistent coding standards, improving the correctness and maintainability of our codebase [18].

FUTURE OF THE SCRIPT GENERATOR

Although the script generator can be installed on any machine [19] we are current only deploying releases to two muon instruments [20] [21], for which we have met the requirements.

We are now looking at expanding the script generator functionality to make it useful to other instruments [22]. The project is being managed as a subproject of the IBEX project and fits into the scrum methodology used for the IBEX project [23]. Scrum requires work packages to be split into manageable chunks taken from requirements.

A new requirements document is being created to cover all interested parties. These requirements are being split into milestones in order to provide a roadmap to scientists and developers. The roadmap can then be used when planning work for the project.

The requirements are also split into the following categories: dynamic scripting, estimation, generated scripts, actions and parameters, gestures, script definitions, usability and miscellaneous.

Dynamic scripting captures behaviours which interact with the script server to run and control scripts directly from the script generator. Eventually a user will be able to start, stop and pause execution of the script, edit actions whilst the script is running, and execute dry runs.

Dry runs will not interact with any hardware but use a simulated version of genie python to test scripts act as expected. The script generator will display feedback from the running script to ensure the user knows the status of their running script.

Time estimation of a script is achievable with the current script generator, but scientists would also like to be able to estimate other values. For example, muon instrument scientists want to display estimated events. This is captured in the estimation category as well as displaying an estimate of when the script would finish if run at the current time.

There is some debate as to the best method of saving and reloading parameters into the script generator. Using a second file to store the parameters is confusing to users. Work to hide or remove the second file, as well as improving the readability of generated scripts and reloading of global parameters is captured by the generated scripts category.

Currently, the script generator only accepts values in the table as strings. We plan to define types for action parameters. This will enable script definitions to configure displays such as having drop downs for enumerated types and only accepting numbers as inputs for integers. This typing system is captured in the actions and parameters category.

ISBN: 978-3-95450-221-9 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2021-TUPV049

Also captured in this category is the ability to select different script definitions for each action in the table and creation of configurable objects. These objects are passed a bit like global parameters through the entire script but are decoupled from script definitions so can be used in different types of actions. Both the configurable objects and ability to select different script definitions for each action are inspired by another script generator currently in development at ISIS.

The gestures category captures the ability to export and import parameters to and from excel spreadsheets and looping more intelligently over actions.

We want to enable ease of development for script definitions to make the script generator a tool of choice for instrument scientists. Instrument scientists may not have experience in tools to manage code bases and so we have devised a workflow for managing them easily which is captured in the script definitions category.

The usability category defines work to improve the user experience, such as displaying more helpful errors when a script definition is invalid, improving the visibility of problems on the loading screen, and a UI review and subsequent redesign.

Finally, the miscellaneous category captures a few smaller work packets that do not fit into the categories listed previously, such as bugs to fix, some reorganising of where we store our script generator python code and reducing the size of the built script generator.

CONCLUSION

The IBEX script generator fulfils many of the required behaviours to improve the reliability of scripting experiments and is well on the way to being a useful tool for beamlines at the ISIS Neutron and Muon Source. However, more work is required to fulfil the requirements of those beamlines and improve the usability of the script generator.

Developing the script generator has come with a number of challenges including using a bridge between Java and Python. These challenges and the object-oriented design techniques employed have helped form the software architecture of the script generator.

We utilise several quality assurance measures to give us confidence in the functionality of the script generator. Although these measures have not caught all bugs and usability issues they have caught a number of significant problems. Our iterative approach and close working with a small number of early users have allowed us to pre-emptively identify issues and refine the script generator to improve its robustness.

Development work will continue to run within the IBEX project. A renewed focus to work with a wider user base will help us to develop a roadmap to fulfil the requirements of more beamlines at ISIS and drive future work on the script generator.

REFERENCES

[1] F.A. Akeroyd *et al.*, "IBEX - the New EPICS Based Instrument Control System at the ISIS Pulsed Neutron and Muon Source", in Proc. 15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15), Melbourne, Australia, October 2015, paper MOPGF048, pp. 205-208, doi:10.18429/JACoW-ICALEPCS2015-MOPGF048

[2] Genie Python, https://github.com/ISISComputingGroup/genie_python

[3] Experiment controls group (ISIS), https://www.isis.stfc.ac.uk/Pages/Experiment-Control.aspx

[4] Nicos script server, https://nicos-controls.org/

[5] J. McAffer, J. Lemieux, and C. Aniszczyk. "Eclipse rich client platform", Addison-Wesley Professional, 2010.

[6] E. Freeman, *et al.*, "Head First Design Patterns", O'Reilly Media, Inc., 2008.

[7] M. Labanda-Jaramillo *et al.*, "Empirical Study Between Compiled, Interpreted, and Dynamic Programming Languages Applying Stable Ordering Algorithms (Case Study: Java, Python, Jython, Jpype and Py4J)", *KnE Engineering*, 2018, pp. 122-132.

[8] V. Gaudioso "Mvvm: Model-view-viewmodel", in *Foundation Expression Blend 4 with Silverlight*, Apress, 2010, pp. 341-367.

[9] CompletableFuture mechanism, https://www.baeldung.com/java-completablefuture

[10] JUnit, https://junit.org/junit5/

[11] unittest, https://docs.python.org/3/library/unittest.html

[12] Jenkins, https://www.jenkins.io/

[13] Squish, https://www.froglogic.com/squish/

[14] D. North, "Introducing bdd", *Better Software,* vol. 12, 2006.

[15] Gherkin language, https://cucumber.io/docs/gherkin/

[16] Squish BDD Tools, https://www.froglogic.com/squish/features/bdd-behavior-driven-development-testing/

[17] Checkstyle, https://checkstyle.sourceforge.io/

[18] S. Edwards, N. Kandru, and BM. Rajopal, "Investigating static analysis errors in student Java programs", in *Proc. 2017 ACM Conference on International Computing Education Research*, Tacoma, WA, USA, Aug. 2017, pp. 65-73.

[19] Installing the IBEX Script Generator, https://github.com/ISISComputingGroup/ibex_user_manual/wiki/Downloading-and-Installing-The-IBEX-Script-Generator

[20] EMU Beamline, https://www.isis.stfc.ac.uk/Pages/emu.aspx

[21] MuSR Beamline, https://www.isis.stfc.ac.uk/Pages/musr.aspx

[22] ISIS Beamlines, https://www.isis.stfc.ac.uk/Pages/Instruments.aspx

[23] K. Baker *et al.*, "Agility in Managing Experiment Control Software Systems", presented at ICALEPCS'21, Shanghai, China, Oct. 2021, paper WEAR03, this conference.