COMPUTING SCIENCE

Reminiscences of Whetstone ALGOL

Brian Randell

TECHNICAL REPORT SERIES

No. CS-TR-1190 **February, 2010**

TECHNICAL REPORT SERIES

No. CS-TR-1190 February, 2010

Reminiscences of Whetstone ALGOL

B. Randell

Abstract

These reminiscences centre on the implementation, by Lawford Russell and myself, of an ALGOL 60 compiler for the English Electric KDF9 Computer in the early 1960s. However details are also given of preceding work on so-called "automatic programming", and of other contemporary ALGOL compiler projects.

© 2010 University of Newcastle upon Tyne.
Printed and published by the University of Newcastle upon Tyne,
Computing Science, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.

Bibliographical details

RANDELL, B

Reminisences of Whetstone ALGOL
[By] B. Randell

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Science, 2010.

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-1190)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE
Computing Science. Technical Report Series. CS-TR-1190

Abstract

These reminiscences centre on the implementation, by Lawford Russell and myself, of an ALGOL 60 compiler for the English Electric KDF9 Computer in the early 1960s. However details are also given of preceding work on so-called "automatic programming", and of other contemporary ALGOL compiler projects.

About the author

Brian's earliest work, during the period 1957-1964 while he was at English Electric, was on compilers. This led to the book: *Algol 60 Implementation*. (Co-author L. J. Russell). Academic Press, London, 1964. He then joined IBM T.J. Watson Research Center, Yorktown Heights, N.Y. where, with an intervening year during 1965-66 in California, he worked on high performance computer architectures (the ACS Project), then on operating systems and system design methodology. During this time, and shortly after he returned to the UK to become Professor of Computing Science at the University of Newcastle upon Tyne. He was co-editor of the reports on the two NATO Software Engineering Conferences. In 1971 Brian set up the project that initiated research into the possibility of software fault tolerance, and introduced the "recovery block" concept. Subsequent major developments included the Newcastle Connection, and the prototype distributed Secure System. He has been Principal Investigator on a succession of research projects on system dependability funded by the Science Research Council (now Engineering and Physical Sciences Research Council), the Ministry of Defence, the European Strategic Programme of Research in Information Technology (ESPRIT), and the European Information Society Technologies (IST) Programme. Most recently Brian has performed the role of Project Director for CaberNet (the IST Network of Excellence on Distributed Computing Systems Architectures) and for two IST Research Projects, MAFTIA (Malicious- and Accidental-Fault Tolerance for Internet Applications) and DSoS (Dependable Systems of Systems). Subsequently he was involved with the RODIN IST project, and the ReSIST IST Network of Excellence. Brian's current computing science research continues to be focused on Dependability (for example on failure analysis) and, to a lesser extent, on the History of Computing.

Suggested keywords

COMPILERS
KDF9
ALGOL 60
E.W. DIJKSTRA

Reminiscences of Whetstone ALGOL

The 50th Anniversary of the Publication of the ALGOL 60 report

BCS Advanced Programming Specialist Group
and the
Computer Conservation Society

Thursday 14th January 2010

Brian Randell <brian.randell@ncl.ac.uk>

Newcastle University

I was hired from college in 1957 by English Electric to program nuclear reactor codes on DEUCE. This was at their Atomic Power Division in Whetstone, just outside Leicester. However, I first spent the summer with IBM at their main London offices, which were in Wigmore Street, behind Selfridges Department Store.

At IBM I programmed and operated a 650 computer, at the time IBM's only computer in the UK. Their 650 was proudly installed behind large plate glass windows, where it attracted great attention from the passers by. After the sleek lines of the 650, with its simple and straightforward programming facilities, and its glossy printed documentation, the DEUCE's workmanlike cabinetry, its binary-level programming, and its duplicated and stapled typewritten manuals, came as rather a shock to me when I arrived at Whetstone in the autumn of 1957.

The DEUCE, English Electric's first computer, was largely based on Alan Turing's 1945, i.e. post-Bletchley Park, plans at NPL for the ACE computer [Turing 1945]. It therefore differed greatly from just about all its contemporaries, both British and American. The ancestry of these other machines could all be traced back more or less directly to the work of von Neumann and his colleagues on the design for EDVAC at the Moore School of the University of Pennsylvania [von Neumann 1945]. In particular, DEUCE's design emphasised computation speed and hardware economy. The assumption was that programmers would be willing and able to take full advantage of the machine's very low-level form of programming. This involved working in binary, and directly controlling the timing of bringing operands to function units and of initiating the operation of these units – in effect a form of what later came to be known as microprogramming. The effect was that, in the hands of an expert programmer, DEUCE could considerably outperform other computers of basically similar hardware speed.

I and a fellow maths graduate from Imperial College, Mike Kelly, who joined Whetstone at the same time as me, soon came to regard DEUCE with affection, even as a giant toy. So we did not confine our programming to nuclear reactor codes, but instead, on our own initiative, wrote numerous demonstration programs, and

started investigating how we could get DEUCE to help with some of the more detailed clerical aspects of its own programming.

DEUCE was in fact not only difficult to program, it was also very difficult to compile for. Most of the then-existing aids to programming DEUCE were interpreters rather than compilers. The effectiveness of these interpreters depended on the cost of the interpretation loop relative to the value of the individual interpreted operations. Thus NPL's interpreter controlling execution of a set of (efficiently hand-coded) linear algebra routines was very cost effective, and being also convenient was heavily used [Robinson 1960]. At the other end of the spectrum was Alphacode, an interpreter that had been developed for a simple language for floating point scalar arithmetic [Denison 1960]. This was normally used only for small one-off calculations since programs written in it were typically orders of magnitude slower than the equivalent hand-optimised machine code.

The fact that DEUCE, like all the earliest machines, did not have any built-in floating-point arithmetic was very significant. There were subroutines for the various basic floating-point arithmetic operations. But in the interests of efficiency many calculations were laboriously programmed using fixed-point arithmetic. Indeed we programmers regarded the use of floating point almost as cheating, and the coming of hardware floating point as more of a challenge than an opportunity. Against this background Mike and I came up with a compromise solution that took advantage of some peculiar DEUCE hardware features in order to produce a very fast interpreter, albeit just for fixed-point computations. (As I recall it, the main loop of our interpreter had only five instructions, whereas more conventional interpreters typically employed loops involving fifty to a hundred instructions.) Despite management opposition – and indeed threats of dismissal – we pressed on with our ideas, and mainly in our own time developed a system, which we called EASICODE [Kelly and Randell 1960, 1962].

EASICODE was so successful, particularly among the scientists and engineers who were the main computer users at Whetstone, that by the time the KDF9 came onto the horizon, I was in charge of a small "Automatic Programming Department". Lawford Russell had joined me, replacing Mike Kelly who had left for pastures new at IBM (where he put his great coding skills to developing the microcode for what became the System 360/40). Moreover, there was full management support for us to exercise our skills and enthusiasms on this exciting new computer.

One important aspect of our environment was that we were working in very close proximity to the users of our programs, and had extensive experience of operating our own and other people's programs on the DEUCE. We therefore had developed strong (self-defensive) views on the need for programs to be robust – in the face of input, operator and hardware errors, and for them to provide meaningful feedback to users whenever possible. (Little did I know that such experiences and views would strongly influence most of my subsequent career, not just our plans for KDF9.)

KDF9 was a floating point machine, with a set of 16 high speed registers, organised as a small arithmetic stack or pushdown store, together with another set acting as a pushdown store for subroutine return addresses. So it was very different from DEUCE, and indeed all other computers. (Superficially it had similarities with the contemporary Burroughs B5000, but in fact it presented a very different and harder challenge to compiler writers than the B5000.) Indeed, as with DEUCE, carefully hand-coded KDF9 programs could be very efficient, but the problem of compiling comparably good code from a high-level language was far from straightforward.

Faced with this challenge, and a market where IBM's scientific computers already had an effective optimising compiler for FORTRAN, English Electric's Computer Division, at Kidsgrove were planning a very ambitious full optimising compiler [Hawkins and Huxtable 1963] for the recently defined ALGOL 60. It was evident to us that this compiler would not be ready until some considerable time after the first KDF9s were due to become operational. And our view was that, though this compiler might produce fast code, it itself was likely to be far from fast. We therefore decided to concentrate on the early provision of a programming facility that would be suitable for repeated use during program development.

Then Lawford Russell and I, and members of the Kidsgrove team, attended a workshop at the Brighton College of Technology in 1961, organised by their Automatic Programming Information Centre. At this meeting Edsger Dijkstra described the then very new ALGOL 60 compiler for the Electrologica X1 computer that he and his colleague Jaap A. Zonneveld had implemented [Dijkstra 1963]. At this stage Lawford and I had not decided what programming language we should support, and were showing dangerous signs of wanting to create our own. Luckily, one of the Kidsgrove group suggested that we should approach Dijkstra to find out if he would support our basing our work on his compiler. He readily agreed, and – with some difficulty – we got English Electric to send us to Amsterdam for a week to confer with him.

Our week of discussions with Dijkstra were spent not just on learning how the X1 compiler worked, but also on the design of a high speed translator/interpreter of full ALGOL 60 for the KDF9. These discussions we documented in a lengthy report language [Randell and Russell 1962]. For the next few years Dijkstra used our report to defend himself from the numerous further requests he was getting from people who wanted to visit him and find out about the X1 compiler. (Having lost my own copy of this report many years ago, it was only recently that I found another one after a long search through numerous libraries and archives, and made it available on the Internet. I have also only just been kindly alerted – by Doaitse Swierstra – to the existence of a Report by F.E.J. Kruseman Aretz published in 2003 by the Centrum voor Wiskunde en Informatica, successor to the Mathematical Centre. This report contains the full assembly code text of the X1 Compiler together with that of an equivalent Pascal version [Kruseman Aretz 2003].)

The X1 compiler was I believe the world's first ALGOL compiler, certainly the first to cope with all the major new challenges of ALGOL 60, such as block structuring, call by name parameters, recursion, etc. (Indeed I recall that some of the German

members of the ALGOL Committee, who were struggling to implement the new language that they had voted for, campaigned for the removal of some of these facilities from the language.)

There was at this time very little published literature on the subject of ALGOL compilation. I recall there being just one book on compilers, that by Halstead on compiling NELIAC, a dialect of ALGOL 58 language [Halstead 1962]. (ALGOL 58 introduced a number of the features that were extended and generalized in ALGOL 60 – for example it allowed nesting of procedures, with corresponding scoping of identifiers, but it did not have block structuring.) There had, in addition, been a recent growth in publications on formal syntax and syntax analysis (in large part spurred by the ALGOL 60 Report's use of what became known as BNF), and on arithmetic expression translation. But we relied almost entirely on information and advice that we received from Dijkstra.

We set to, and started the detailed design of what became known as the Whetstone ALGOL Translator. We never used the term WALGOL ourselves, and avoided the word "compiler" because we were converting ALGOL 60 into an intermediate language, of the form that later became known as P-code, rather than into machine instructions. In effect our intermediate language defined the order code of the machine that we would have liked to have had as our target – an order code that was, I learned some years later, one of the sources of inspiration for the Burroughs 6500 computer, the successor to their B5000.

There was considerable, in general friendly, rivalry between Whetstone and Kidsgrove, but also – thanks largely to Fraser Duncan, to whom the Kidsgrove team reported – good cooperation in ensuring the compatibility or our respective systems. In any arguments over such issues we used the ALGOL 60 Report as a neutral referee, so-to-speak, and thus both projects stuck pretty closely to the letter of the Report.

We at Whetstone placed considerable emphasis on easing the program development task. One consequence was that we were dissatisfied by Dijkstra's strategy of having his compiler signal the first error that it found and then refuse to carry on any further – rather we worked hard on developing a strategy whereby our translator would process the entirety of even a very error-ridden program. It did this in such a way as to have a very good chance of producing an accurate list of many of the errors contained in the program, without getting confused and reporting lots of alleged errors in perfectly good sections of program text. I still recall that when Dijkstra wrote to acknowledge the copy of the report that we had sent him describing this scheme he for the first time addressed me as "Brian". In all our extensive previous correspondence he had always addressed me as "Mr Randell". I felt as though I'd had just been awarded a Masters degree!

We chose to invent a flow diagram notation to use for the detailed design and documentation of our translator and interpreter. (The idea of documenting our design in ALGOL and using bootstrapping as way a means of implementing it, an approach used for NELIAC, was too strange and unfamiliar an idea for us to

contemplate at all seriously.) We then hand-coded the flow diagrams in the KDF9's assembly language ("usercode") and used a painfully slow emulator of the KDF9 that had been written for DEUCE to start the process of checking them.

But then the delivery date for the first KDF9 started slipping – just a few weeks at a time, but over and over again. These delays were very disruptive, but in fact we ended up putting them to very good use. Someone suggested that we publish the details of Whetstone ALGOL in book form. We sought Dijkstra's views on this. He was very supportive, and gave me some very good advice, that I've since passed on to many graduate students. This was to the effect that a straightforward description of our system would be of interest to only a very limited readership. However, an account that documented all the possibilities we had considered for each design decision that we had taken, explained why we had made our particular design choices (admitting where necessary that a decision had been arbitrary), and reviewed the merits of these decisions in the light of our subsequent experience, would be of much greater value.

So we set out to write such a book, while we awaited the arrival of the KDF9. We worked on the book very intensively, since we had no idea how long the delays would continue, and we thought (in fact wrongly) that others elsewhere, in particular Peter Naur in Copenhagen and Gerrit van der Mey in at the Dr. Neher Laboratorium in the Netherlands, were already busy preparing books on their ALGOL efforts. We completed the book in nine months, mainly in our spare time.

Our inexperience as authors resulted in our providing the publishers, Academic Press, with a much more polished manuscript than they were used to receiving. We were mindful of all the horror stories that Dijkstra had told us of errors that had been introduced into some of his publications by the manual typesetting process – indeed he urged us to insist on direct photographic reproduction of our typescript. We did not follow this advice, but instead had several typewriters modified so as to have the full ALGOL 60 character set, even including the small subscript "10" character. And we also had all numeric digits use an italic font – so that there could be no confusion between the letter "O" and the digit "*0*", or between the letter "l" and the digit "*1*". These precautions paid off handsomely.

Somewhere during this process, reacting to an announcement of yet another two-week delay, for fun we started to see what we could produce during this period in the way of a little ALGOL-like system for DEUCE, with one-character identifiers (associated with specific words in a 32-word "mercury delay line" memory). A small succession of further delays, and consequent extensions of our little toy system, ended up with our having to - our own surprise - produced a useful system. This implemented block structure, simple arithmetic expressions, assignment and for statements, procedures and simple parameters, and input/output. So quite a few people started using it in anger for small one-off calculations. But once the first KDF9 became operational at Kidsgrove we abandoned this diversion.

Meanwhile various colleagues, at Whetstone and elsewhere, started to implement our flow diagrams in order to provide full ALGOL 60 on various other machines,

including even DEUCE. In so doing they provided valuable feedback on both the diagrams, and the early drafts of our book. (I recall that the chapter on blocks and display, at the time very novel and difficult to understand concepts, was rewritten at least five times. It ended up as perhaps the clearest part of the book!) As a result, when we at last got access to a more-or-less working KDF9 our implementation of Whetstone ALGOL went very smoothly. (For a while, if we ever had unexpected results when running our system on the first KDF9, it was our habit to try to reproduce them on Kidsgrove's DEUCE emulator for KDF9. Any time the results differed we could be sure that the machine and/or the emulator were at fault, and we would gleefully hand both back to their developers while they found out which needed to be mended.)

Our system was designed to work on a minimum configuration KDF9, with 8k word memory, and to read its input directly from paper tape. Given its intended use as a program development system, we put considerable effort into ensuring that the translator operated in a single pass. And Lawford did such a good job of overlapping translation with input that the translator kept the paper tape reader operating at full speed, and there was no perceptible delay after the end of tape was reached before the translated program started running. Thus users could regard our system as essentially cost-free. (In contrast, to our great scorn, KDF9's assembler seemed to do most of its work only after reading had finished.)

Another ALGOL compiler activity of which we were directly aware was that by Tony Hoare at Elliott Brothers. His compiler, for the Elliott 803, was in fact I believe operational before any of the compilers based on our design. However, it was for a somewhat limited version of ALGOL. Thus when Tony published his famous QuickSort algorithm in the Communications of the ACM [Hoare 1961], he included a version that avoided the use of recursion, since this was not supported by his compiler. I'm afraid that, as soon as our system was operational, we rather mischievously sent in a certification of the original recursive version of QuickSort, whose publication in effect signalled the completion of Whetstone ALGOL [Randell and Russell 1963].

I have already alluded to the ALGOL compiler designed by Gerrit Van der Mey – probably the most impressive of the early ALGOL compiler efforts [van der Mey 1962]. This was not just because it was the most complete implementation of the language of which I was aware – it even handled dynamic own arrays and various other unfortunate little complications of the language that just about everybody else avoided tackling. Rather it was because van der Mey, who I met just once when I was taken to his house by Fraser Duncan, was both totally blind and deaf. Yet despite these handicaps, he had almost singlehandedly designed his compiler, though colleagues at the Dr. Neher Laboratorium helped with such debugging as was needed. The one other ALGOL Compiler I can recall knowing about in those days was an amazing effort by Brian Higman, for an incredibly small machine, an experimental process control computer being developed by G.E.C. whose total memory capacity was 512 20-bit words [Higman 1963]. But elsewhere in the UK the

emphasis was on Autocode rather than ALGOL, but this was a subject we had very little contact with.

By 1964 the Whetstone System was operational, our book had been published [Randell and Russell 1964], more compiler projects had been undertaken based on it, and I had been head-hunted by IBM to join the T.J. Watson Research Center in Yorktown Heights. I had agreed to join IBM on condition that I was not asked to continue working on ALGOL compilers – I wanted a change. They agreed, and moreover agreed to my accepting the invitation that I'd had a long time earlier to join the IFIP Working Group 2.1 on ALGOL, an invitation that English Electric had not let me accept because of the costs of the foreign travel involved. (The Working Group was formed in 1962 as a successor to the original ALGOL Committee.) So I attended the 1964 meeting of the Committee, and the associated IFIP Working Conference on "Formal Language Description Languages", held in Baden, near Vienna [Steel 1966]. (This conference attempted, not altogether successfully, to bring together people interested in language theory and ones interested in actual programming languages and their compilation.) And thus, over the next few years, I found myself in the midst of the growing controversy over plans for a successor to ALGOL 60, culminating in being one of the small group of committee members who in 1968 wrote a Minority Report on ALGOL 68, and resigned en masse from the Committee. But that is another story.

One final anecdote: when I returned to the UK in 1969, to the Chair of Computing Science at Newcastle University, I found that Whetstone ALGOL was still the mainstay of their first-year programming courses, and so was being used very extensively on their soon-to-be-replaced KDF9. I took care to make it clear that I wanted no part in dealing with any problems that the University might have with the system, only to be told: "We wouldn't let you touch it even if you wanted to!" Similarly, I also can claim no credit at all for the recent splendid effort by David Holdsworth and colleagues that has succeeded in getting the Whetstone ALGOL system working again, starting from a line-printer listing of its code, on an emulator of the KDF9.

References

[Denison 1960] S.J.M. Denison. "Further DEUCE Interpretative Programs and some Translating Programs," in *Annual Review in Automatic Programming (Vol. 1), ed. R. Goodman*, pp. 127-139, Oxford, Pergamon Press, 1960.

[Dijkstra 1963] E.W. Dijkstra. "Making a Translator for ALGOL 60," in *Annual Review in Automatic Programming (Vol. 3), ed. R. Goodman*, pp. 347-356, Oxford, Pergamon Press, 1963.

[Halstead 1962] M.H.A. Halstead. *Machine-Independent Computer Programming,* Spartan Books, 1962.

[Hawkins and Huxtable 1963] E.N. Hawkins and D.H.R. Huxtable. "A Multi-Pass Translation Scheme for ALGOL 60," in *Annual Review in Automatic Programming (Vol. 3), ed. R. Goodman*, pp. 163-205, Oxford, Pergamon Press, 1963.

[Higman 1963] B. Higman. “Towards an ALGOL Translator,” in *Annual Review in Automatic Programming (Vol. 3), ed. R. Goodman*, pp. 121-162, Oxford, Pergamon Press, 1963.

[Hoare 1961] C.A.R. Hoare, “Algorithm 64: Quicksort,” *Comm. ACM*, vol. 4, no. 7, p.321, 1961.

[Kelly and Randell 1958] M.J. Kelly and B. Randell. *Preliminary Report on EASICODE,* W/AT 216, Atomic Power Division, English Electric Co., 1958. http://www.cs.ncl.ac.uk/publications/trnn/papers/133.pdf

[Kelly and Randell 1960] M.J. Kelly and B. Randell. *EASICODE,* W/AT 585, Atomic Power Division, English Electric Co., 1960. http://www.cs.ncl.ac.uk/publications/trnn/papers/32.pdf

[Kruseman Aretz 2003] F.E.J. Kruseman Aretz. *The Dijkstra–Zonneveld ALGOL 60 compiler for the Electrologica X1,* SEN-NO301, Centrum voor Wiskunde en Informatica, 2003.

[Randell and Russell 1962] B. Randell and L.J. Russell. *Discussions on ALGOL Translation, at Mathematisch Centrum,* W/AT 841, Atomic Power Division, English Electric Co., 1962. http://www.cs.ncl.ac.uk/publications/trnn/papers/34.pdf

[Randell and Russell 1963] B. Randell and L.J. Russell, “Certification of Algorithms 63, 64 and 65, Partition, Quicksort, and Find,” *Comm. ACM*, vol. 6, no. 8, pp.446, 1963.

[Randell and Russell 1964] B. Randell and L.J. Russell. *ALGOL 60 Implementation,* London, Academic Press, 1964.

[Robinson 1960] C. Robinson. “Automatic Programming of DEUCE,” in *Annual Review in Automatic Programming (Vol. 3), ed. R. Goodman*, pp. 111-126, Oxford, Pergamon, 1960.

[Steel 1966] T.B. Steel, (Ed.). *Formal Language Description Languages for Computer Programming,* North-Holland Publishing Co., 1966.

[Turing 1945] A.M. Turing. *Proposals for the Development in the Mathematics Division of an Automatic Computing Engine (ACE),* Report E882, National Physical Laboratory, 1945. [Reprinted with foreword by D.W. Davies, NPL Report Comm. Sci. 57, April 1972.]

[van der Mey 1962] G. van der Mey. *Process for an ALGOL Translator,* Dr. Neher Laboratorium, Staatsbedrijf der Posterijen, Telegrafie en Telephonie, 1962.

[von Neumann 1945] J. von Neumann. *First Draft of a Report on the EDVAC. Contract No. W-670-ORD-4926,* Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, PA, 1945. [(Extracts reprinted in *Origins of Digital Computers: Selected Papers* (ed. B. Randell) Springer-Verlag, 1973.)]

Algol /ËˆÃ¦lÉ¡Ê’l/, designated Beta Persei (Î² Persei, abbreviated Beta Per, Î² Per), known colloquially as the Demon Star, is a bright multiple star in the constellation of Perseus and one of the first non-nova variable stars to be discovered. Algol is a three-star system, consisting of Beta Persei Aa1, Aa2, and Ab â€“ in which the hot luminous primary Î² Persei Aa1 and the larger, but cooler and fainter, Î² Persei Aa2 regularly pass in front of each other, causing eclipses. Thus Algol's magnitude is usually Reminiscences of Whetstone Algol. B Randell. Stijlen van programmeren, 1952-1972.Â Current views on the programming language Algol assume its European origins. However, the inability to exchange information between computers affected both sides of the Atlantic. Whereas Algol promoters sought to create one universal programming language, other approaches sought to preserve a variety of languages and create a general translation system. Therefore, the polarity was not between programming languages, but uniformity versus diversity. View. Show abstract. The european side of the last phase of the development of algol 60. Article. Jan 1978. P. Naur. The Whetstone benchmark is a synthetic benchmark for evaluating the performance of computers.[1] It was first written in Algol 60 in 1972 at TSU (The Technical Support Unit of the Department of Trade and Industry - later part of the Central Computer and Telecommunications Agency or CCTA in the United Kingdom). It was derived from statistics on program behaviour gathered on the KDF9 computer at NPL National Physical Laboratory in the United Kingdom, using a modified version of its Whetstone ALGOL 60 compiler.Â The Whetstone Compiler was built at the Atomic Power Division of the English Electric Company in Whetstone, Leicestershire, England,[2] hence its name. Blessed Whetstone is a Common Component for use on Axes, Swords, Guns and Crossbows. It is obtained primarily through crafting. The Blueprint: Blessed Whetstone is available from the Kymon's Chosen and Order of Death's Vigil Faction Quartermasters after reaching Respected status. This Component can also be received as part of Faction Bounty rewards for The Black Legion.